顧客管理システム「IVORY」の開発

021030 田上義高*1　　031211 上田卓也*2　　031371 服部智成*3

*1 システム研究科 3 年 B クラス　*2 情報システム科 2 年 C クラス　*3 インターネットシステム科 2 年

要旨：本研究は、麻生教育サービス株式会社（以下、AES）の要望により顧客管理システムを構築するものである。AES ではプロバイダの代理店業務において、顧客を Microsoft の Excel で管理している。しかし、顧客の人数が膨大なため、この手法では管理に手間がかかっているといった現状がある。そこで、サーバーサイド Java 技術を用いた分散アプリケーションを新たに構築する。

キーワード:Web アプリケーション、顧客管理

1.　開発の経緯

本プロジェクトは顧客管理システム「Ivory」を開発し、麻生教育サービス株式会社様（以下 AES という。敬称略）にて実運用することを目的とするものである。AES ではインターネットサービスプロバイダ（以下プロバイダという）の代理店業務を執り行っており、その業務改善策を検討されていた。一方私たち研究生は、実運用でも通用するシステムの開発を模索していた。双方の考えが一致したため、研究室の取り計らいによって、本システムの開発を担当することとなった。

2.　ヒアリング

2.1.　ヒアリングの実施

開発を開始するに当たり、AES がどのようなシステムを望んでいるかを把握するため、担当者と面会する機会を設けた。ヒアリングの際に専門用語の使用を避けるよう努め、図を用いることで担当者との意思の疎通を図った。そして現状の業務プロセスとシステムに要求される機能を UML 図を用いて記録として残した。

2.2.　顧客管理の現状

AES では、プロバイダの会員情報を Microsoft 社の表計算ソフト Excel（以下 Excel という）で管理している。その件数は 5000 件にも及び、データ変更に加えて課金処理に大変な手間がかかるものであることがわかった。

本件プロバイダにてインターネットを利用する者（以下登録者という）は、個人が直接加入するケースと法人の従業員等が法人を介して加入するケースとに分かれる。利用料の請求は、個人の場合はその本人宛となり、法人の場合は法人宛となる。請求する料金を計算する場合、通常は請求の単位である個人と法人にのみ着目するだけでよい。しかし、インターネットの利用には回線の変更がつきものであって、登録者の利用料の単価は区々となる。そこで請求の単位とは別に登録者の回線情報を把握することが重要となる。

3.　要件定義

AES の要望と業務の現状を踏まえた上で、次のように要件定義を行った。「Ivory」に要求される機能は「会計管理」と「会員管理」とに大別される。

「会計管理」では回線毎の登録者人数を把握させることで利用料算出の支援をするとともに、請求書発行済み・入金確認済みであることを登録・確認する機能を設けた。「会員管理」では登録者の閲覧・追加・変更・削除機能に加えて検索機能を設けた。また、Excel 形式のファイルからデータを投入し、検索結果を Excel 形式のファイルで保存することを可能にした。さらに回線履歴を表示させて随時確認できるようにした。

以上の処理の他に法人の追加・削除機能があるが、これは管理者権限でないと操作できない仕様にする。これは AES でのヒアリングの結果、新たに法人を追加・削除することはほとんどないことがわかったためである。通常の業務での誤操作を防止する意味でこうした措置を採った。

4.　開発体制

4.1.　メンバーごとの役割

メンバーごとの役割分担は次の通りである。

田上…チームリーダーとして、プロジェクト全体を統括する。また「Servlet」の開発を担当した。

上田…データベースの開発と画面設計を担当した。

服部…ビジネスロジックの開発を担当した。

4.2.　優先順位

プロジェクトマネジメント上での優先順位の三要素である「品質」「資源」「時間」を用いて、開発の能率を向上させるよう努めた。実際に業務で利用されるシステムとなるため、「品質」を最優先として、「資源」をそれに次ぐ優先度とした。納期を表す「時間」については厳格に取り扱わなければならないが、先の二点を重視したいとの考えから十分に配慮しつつも優先度を下げた。

4.3.　開発環境

今回の研究では開発環境は全て 0 円で用意することにした。だが、全て 0 円という環境も以下のようなメリット、デメリットがある。

メリット

・バージョンのアップダウンも無償であるため、バージョ

ンのアップダウンによる金銭的負担が少ない。

・自由に拡張できる。

・ソースコードを自由に閲覧でき、システム開発の勉強になる。

・ライセンスによっては拡張後再配布したり、販売したりが可能。

デメリット

・サポートされている機能が少ない。

・バグによってデータやプログラムが破壊されても基本的に無保証。

最近ではデータベースで有名な Oracle 社の JDeveloper10g など安価な統合開発環境などが発売され始めているため、今後の開発で検討していこうと思う。

4.3.1. 統合開発環境

統合開発環境には Eclipse を採用した。Eclipse は IBM が巨額の資金をかけて開発したものを非営利組織に寄付したもので、無料無償の統合開発環境である。開発開始当時の最新版は 2.1.3 であったためこのバージョンを使用した。

Eclipse には以下のような機能がある。

・Java のコードアシスト

・Junit を正式サポートし、GUI での結果レポート

・色分けによってコードの理解を深めるエディター

・パースペクティブとビューによる見やすい GUI 環境

・CVS を正式サポートし、チーム開発が容易

・プラットフォームを問わないビルダーである Ant をサポート

Eclipse はプログインを追加することによって様々な機能を追加できる。代表的なものを以下に列挙する。

・Sysdeo Eclipse Tomcat Launcher plugin

Tomcat の起動と停止を行い、起動状況や例外の取得、デバッグなどの機能を実現するプラグイン。

・CheckStyle

コーディング規約を設定するプラグイン。だが、独自のコンパイラを使用するために Tomcat のプラグインとコンパイラが競合した。

・Clay データベースモデリング

データベースの設計を行うためのプラグイン。データ型やテーブル定義などの E-R 図を作成できる。またデータベースごとに違う SQL 方言も定義できる。作成したデータ定義は SQL 文にして吐き出すこともできる。

・MNPZ Eclipse SQL Environment Plug-in

・データベース内のデータなどを遠隔的に閲覧、更新できるプラグイン。だが、文字コードが違うことによってエラーが出たりすることがあったためデータ定義の更新などは極力このプラグインで行わないようにしていた。だが、最近は更新が止まっており Eclipse 最新版にも対応していない。

これらのプラグインを基本として利用していく。だが、個人でいいと思ったプラグインは個人の責任の範囲内で利用していいものとした。

4.3.2. データベース

今回のシステムでは 5000 件というデータ件数であるため、更新や参照が早いデータベースを採用しようと考えていた。参照や更新を可能な限り早くするため、ストアドプロシージャを利用したいと考えていた。だが無償データベースでストアドプロシージャを実装しているデータベースは少なく、その中でドキュメントが豊富なデータベースは PostgreSQL しかなかったため採用した。

今回のデータベースのストアドプロシージャは CVS によりバージョン管理し、チーム全員でバグの特定などを行っている。

4.3.3. チーム規約

今回の開発では、お互い共通の決まりなどを設けた方が意思の疎通などの面で円滑に開発が進められると考えたのでチーム規約を策定した。チーム規約では、チーム間で望ましくないことやチーム全員が使うような CVS の使い方などを策定した。それらは全部で 11 個に分類され、策定後もチームで意見があれば話し合いの末、改変や新たな規約の追加も可能にした。

5. 会員管理

会員管理は、「法人」、「個人会員」、「登録者」の情報を管理するものである。

本システムでは、以下の機能を実装した。

5.1. 法人の変更

「法人」のデータを変更する。

法人の名称・住所・料金体系等が変更になる場合があるため、この機能を設けた。

5.2. 法人の削除

「法人」を削除する。

業務として発生する可能性がため、この機能を設けた。

5.3. 法人のデータ表示

「法人」の詳細なデータを表示する。

請求書の発行時に必要となるため、この機能を設けた。

5.4. 登録者の追加

「登録者」を追加する。

学校であれば新入生。会社であれば新入社員など、新たに発生した「登録者」を追加する処理が必要ため、この機能を設けた。

法人・ぷららとの新規登録者のデータの受け渡しは Excel で行うため、CSV ファイルを使用したデータの一括入力を行えるようにした。

5.5. 登録者の変更

「登録者」のデータを変更する。

登録者の氏名、住所、回線等が変更になる場合があるため、この機能を設けた。

ISDN を ADSL にするなどの回線変更を行った結果、請求が必要になる場合は請求進捗表示と連動させる。

5.6. 登録者の削除

「登録者」を削除する。

ただし、削除した登録者のデータ表示(8)や、復帰(9)ができるよう、データベースからは削除しない。

5.7. 現在利用中の登録者の検索・データ表示

現在利用中の(削除していない)登録者のデータを表示する。

氏名・登録者番号・住所等で検索することが可能。

5.8. 削除した登録者の検索・データ表示

削除した登録者のデータを表示する。

同様に、氏名・登録者番号・住所などで検索することが可能。

5.9. 削除した登録者の復帰

削除した登録者を「削除しなかった」ことにして、登録者を復帰する。

本来は削除した事実をデータベース上で残すべきであるが、データベースの大幅な修正が必要になるため今回は見送った。

5.10. 登録者の回線履歴表示

利用中・削除後にかかわりなく、登録者の回線履歴を表示する。

5.11. 登録者データの CSV 出力

登録者データを CSV 形式で出力する。

6. 会計管理

会計管理は、法人・個人会員に対して請求書を発行するためのデータの表示や、

各種請求に対する請求書発行済み・入金済みのチェックを行うものである。

本システムでは、以下の機能を実装した。

6.1. 請求進捗状況の表示

定期的な請求、随時請求を区分に関係なくまとめて表示し、各請求に対し「請求済み」・「入金済み」を表示する。

6.2. 請求進捗状況の変更

「請求済み」・「入金済み」の状態を変更する。

請求進捗状況表示はチェックボックスにしており、画面から直接変更が可能となっている。

6.3. 登録者数の表示(法人のみ)

登録者数の推移を、現時点での累計と、月別に分けて表示する。

月別の情報は、表示する月を利用者が選択可能である。

累計情報

・累計登録者数(現在までに一度でも登録したことがある登録者の数)

・累計削除者数(現時点で削除している登録者の数)

・利用者数(累計登録者数 － 累計削除者数)

・月別情報

新規登録者数(選択された月に利用開始となる登録者の数)

・削除者数(選択された月に削除した登録者の数)

・月末累計登録者数(月末時点での累計登録者数)

・月末累計削除者数(月末時点での累計削除者数)

・月末利用者数(月末時点での利用者数)

7. 管理者権限

IVORY では利用者が使うとシステムに不具合が発生するような操作や、利用者がほとんど使わないような操作は管理者が行うことによって、利用者によるミスなどを軽減する。

管理者権限で行う操作は「法人の追加」「法人の削除」がある。

7.1. 法人の追加

要件では法人が追加されることはないということだったので、システムでは実装しないことにした。

だが法人の追加はシステムの初期化の際にどうしても必要であるため、管理者権限で実装することにした。

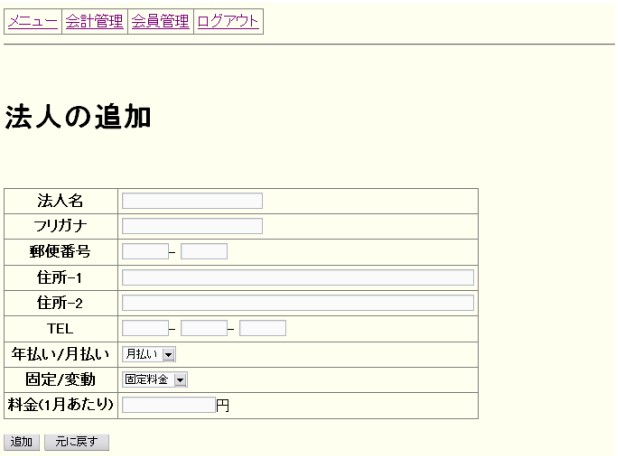


図1 法人追加画面

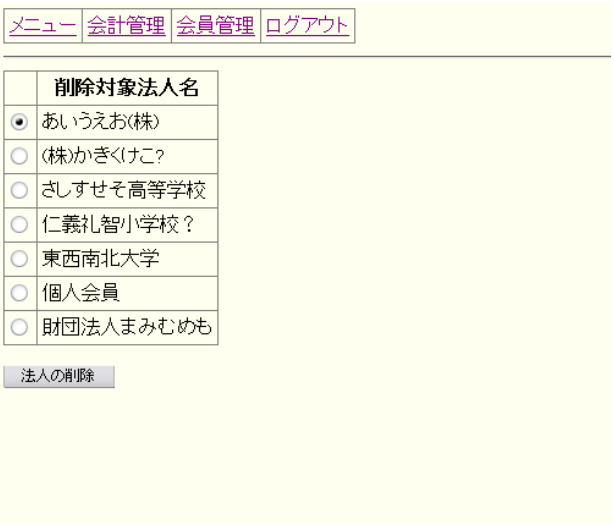


図2 法人削除画面

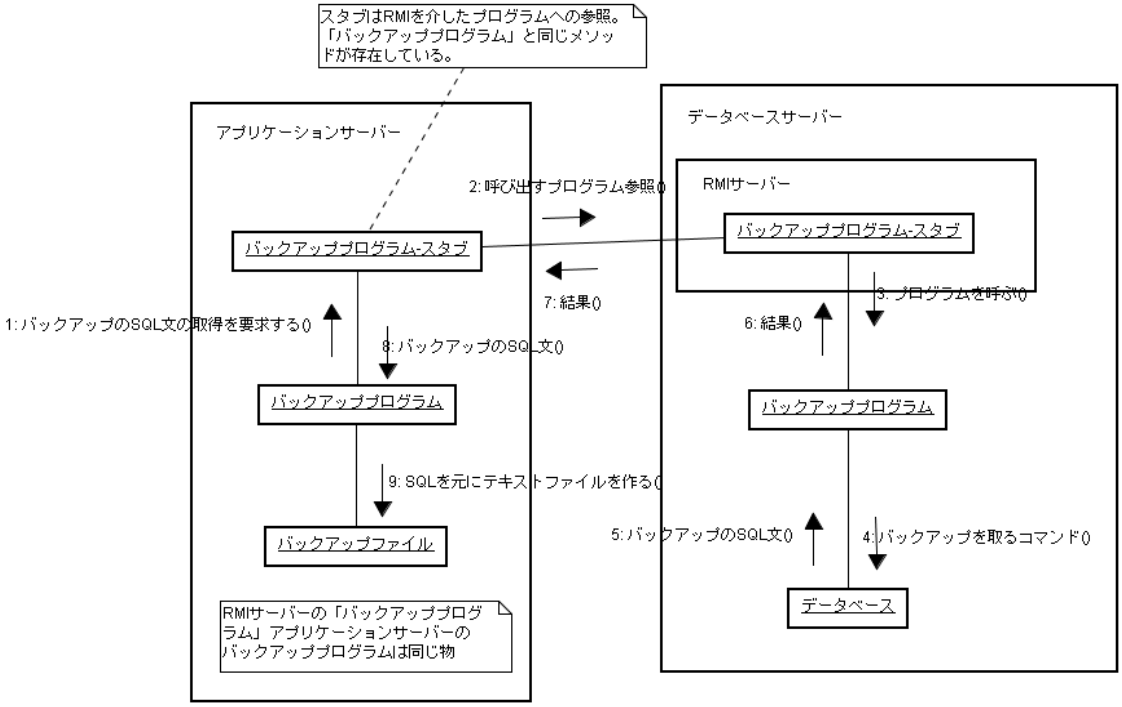


図 3　新しいバックアップの構造

7.2.　法人の削除

要件では法人が削除されることはほぼないということだった。それにシステムでは法人の削除によって法人に所属する全ての登録者が削除されてしまうなどの問題が発生するため、法人の削除は利用者が利用できない部分に実装するのが適当であると考えた。

7.3.　バックアップとリカバリ

バックアップは PostgreSQL のバックアップ機能を用いて実現した。バックアップは Linux のデーモンである Cron を使って定期実行することによって、システム内で毎日バックアップを取っている。

だが、この方法ではシステムが災害や不注意などで物理的に破壊されてしまった場合、データが復旧できない。これを回避するために物理的に破壊されても、データが復旧できるようなシステムの構造を考えた。

図 3 は、たとえマシンが物理的に離れていても自動的、手動的にバックアップが取れるような構成である。まず、データベースサーバーにバックアッププログラムを埋め込んでおき、そのプログラムをアプリケーションサーバー、あるいはユーザーが呼び出すことによって物理的に離れたマシンであってもバックアップを取れるようになる。

これを実現するためにデータベースサーバーに RMI という JavaAPI を使う。RMI は別マシンにあらかじめ用意されているプログラムを呼び出すことによって別マシンの情報を呼び出すことのできる API である。

8.　内部設計

8.1.　MVC モデル

MVC モデルとはシステムの構造を Model、View、Controller という 3 つの概念に分割し、それぞれの役割で分けることによってモジュールの独立性を確保する J2EE デザインパターンである。

Model の役割はシステム内で汎用的に利用するモジュールを定義する。値チェックやデータベースアクセスといったものがある。

View の役割は画面情報の定義、システム内に保存されているデータの参照など、画面ロジックを記述する。

Controller の役割はブラウザから送られてくる、HTTP 要求を受け取り、パラメータや通信情報を制御して HTTP 返送としてブラウザに返す役割がある。

8.2.　Model

8.2.1.　総括

Model の実装には、デザインパターンと既存のライブラリを用いることにした。

8.2.1.1.　デザインパターンの活用

本システムは実際の業務で利用されることを考慮して、容易に機能拡張できることを目指している。そのためデザインパターンを取り入れることで、Model 全体の見通しが効くよう努めた。今回デザインパターンを導入したものはデータベースへのアクセス方法を提供するロジックである。詳細はデータベースアクセスの項目に譲る。

8.2.1.2.　ライブラリの利用

システムの機能を実装する際にそのすべてを一から作成し

ていくことは、限られた時間内で行われる開発では効率が悪いものである。すでに完成して利用実績があるライブラリを利用すると、コーディング作業を必要なものだけに注力でき、ひいてはシステム全体の品質向上につながる。そこで、Jakarta プロジェクト等で提供されているフリーのライブラリを活用することにした（http://jakarta.apache.org/）。

8.2.2. データベースアクセス

データベースにアクセスするためにはJDBC ドライバとデータベースへの接続方法を決定しなければならない。

8.2.2.1. JDBC ドライバの準備

データベースのデータの操作はSQL 文を発行して行う。これを Java のプログラムから行うにはJDBC が必要になる。JDBC は Java Database Connectivity の略で、JDK のコア

```
Class.forName("org.postgresql.Driver");
Connection con = DriverManager.getConnection(
"jdbc:postgresql:db_1※", "postgres","");
※db_1 は任意のデータベース名。
```

図 4「DriverManager」を用いたデータベース接続

API としてパッケージ「java.sql」に実装されている。JDBC はJDBC ドライバマネージャとJDBC ドライバに分けることで、DBMS ベンダに依存しない構成となっている。前者はアプリケーションと JDBC ドライバとの接続を行い、後者は DBMS との接続を行う。

8.2.2.2. データベースへの接続方法

Java からデータベースに接続するには、まず「Connection」オブジェクトを生成する必要がある。この方法として「DriverManager」を用いるものと「DataSource」を用いるものとがある。

「DriverManager」を用いる方法（図「DriverManager」を用いたデータベース接続）はソースコードの再利用性を低下させるため推奨されていない。この方法では、ドライバ名やデータベースにアクセスするために使用される URL がハードコードされる。これらはデータベースに依存するものであるため、将来データベースを変更する必要があるときにはソースコードの修正と再コンパイルが必要となる。

「DataSource」を用いるとその必要がない。

また「DataSource」を用いると、コネクションプーリング機能を利用することができる。この機能の利点は「Connection」オブジェクトを予め指定した数だけ生成して、これを再使用することができる点である。「Connection」オブジェクトをデータベースへの接続の度に生成することは

```
<Context path="/IVORY" reloadable="true"
docBase="" workDir="">
  <Resource name="jdbc/res_name"
    auth="Container"
type="org.postgresql.jdbc3.Jdbc3PoolingDataSource" />
  <ResourceParams name="jdbc/res_name">
    <parameter>
      <name>factory</name>
      <value>
 org.postgresql.jdbc3.Jdbc3ObjectFactory
 </value>
    </parameter>
   …<中略>…
    <parameter>
      <name>maxActive</name>
      <value>100</value>
    </parameter>
    <parameter>
      <name>maxIdle</name>
      <value>30</value>
    </parameter>
    <parameter>
      <name>maxWait</name>
      <value>10000</value>
    </parameter>
  </ResourceParams>
</Context>
```

図 6 「server.xml」の設定

JVM(JavaVirtualMachine)の処理速度の低下を招く。またデータベースへの接続動作頻度が高いとデータベースサーバへの負荷が問題となる。そのため、予めデータベースとの接続を確立した「Connection」オブジェクトをプールと呼ばれる場所に保持させる。利用する際に取り出して、利用が終わるとプールに戻すだけで済むため、システム全体の負荷を低減させるものとして有用なものである。

「DataSource」とコネクションプールはともに「server.xml」で設定することで利用可能となる。

8.2.2.3. 設計手法

データベースへアクセスするためのクラス設計にはデザインパターンを用いることで、運用後のメンテナンス性に配慮した構成に仕上げた。

8.2.2.3.1. 「DataAccessObject」パターン

データベースを将来変更することになった場合、これまで利用してきたJDBCプログラムを新しく導入したデータベースに合わせて変更する必要がある。開発の際にビジネスロジック部分にJDBCプログラムが混在していると、改変作業の範囲が広がりメンテナンス性の低下を招く。

これを避けるため、JDBCプログラムを実装するクラス（以下、DataAcceessObject）にはビジネスロジック部分を実装させない。そして、「DataAccessObject」をその種類に関係なく生成するメソッドを備えたフレームワーク（本パターン）を準備することによって、データベースに対応した「DataAccessObject」を用意するだけでこの種の問題を解決することができる。本パターンを実現するために、「AbstractFactory」、「FactoryMethod」の各パターンを利用した。

8.2.2.3.2. 「AbstractFactory」パターン

「DataAccessObject」を生成するメソッドは、データベースの種類に依存することになるため、このメソッドはフレームワーク中には記述しない。その代わりフレームワーク内には、同一シグニチャの抽象メソッドを記述したクラスを作って、そのサブクラスで実際に「DataAccessObject」を生成させている。ここで登場するスーパークラスは「DataAccessObject」という製品を製造する工場(Factory)に例えることができる。もっともできあがる「DataAccessObject」の種類まで把握するものではなく、単に「何らかの『DataAccessObject』を製造する工場」としての役割を持っているに過ぎない。本パターンの名称に「Abstract」とあるのもその役割が具体的に明らかでないことに由来している。

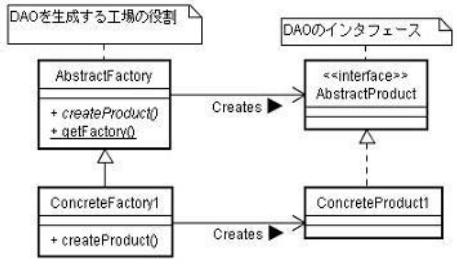


図 8　AbstractFactory

```
InitialContext ic = new InitialContext();
DataSource ds = (DataSource)ic.lookup(
"java:comp/env/jdbc/res_name※");
Connection con = ds.getConnection();
```

図 5 「DataSource」を用いたデータベース接続

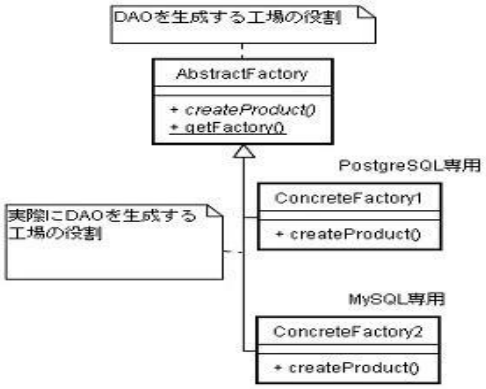


図 7　DataAccessObject

8.2.2.3.3. 「FactoryMethod」パターン

上記フレームワークを利用するときは、抽象メソッドが生成するDAOがどのデータベースに対応しているかを隠蔽する必要がある。そのため「DataAccessObject」には、データベースへアクセスする抽象メソッドを持ったインタフェースを実装させる。本パターンはフレームワーク中の抽象メソッドがインスタンスの生成をサブクラスに委譲するものである。一方「AbstractFactory」はフレームワーク中の抽象クラス（上記の「工場」）に着目している。

8.2.3. ｃｓｖファイルの入出力

Webアプリケーションでよく見られるデータ入力の方法は、テキストボックスを設け、これに必要なデータを書き込んだ後「送信」ボタンを押すものである。入力件数が少ない場合にはこの方法で対処できるが、大量のデータ投入には適さない。すでにファイル形式等でデータが準備できている場合、これをそのまま利用できる方法を導入すると処理の簡素化に役立てられる。

私たちの開発では投入されるデータが予めExcelで準備されていたため、Excel形式のデータをシステムに投入できる方法を模索していた。ExcelのデータはCSV形式（Comma Separated Values）と呼ばれるコンマ区切りのデータとして保持されている。そのためコンマで区切られた個々の文字列（以下トークンという）を獲得できれば、それぞれをシステムで処理することが可能となる。この点で、「java.util.StringTokenizer」クラスでは予め設定された区切り文字で文字列を分割することができる。

しかしこの方法では、トークンにコンマが含まれる場合にこれを区切り文字として認識される点で問題がある。これを解決するライブラリ（以下csv.jarという）がWeb上で公開されており(http://www.wakhok.ac.jp/~tomoharu/Java/csv/)、StringTokenizerクラスに変えることができる。利用方法はStringTokenizerクラスと同様で、メソッド「nextToken」を呼び出すごとにトークンを生成することができる。

なおこのライブラリでは大量のデータ群から CSV 形式のファイルを調製することも可能である。この機能はシステムにて絞込み処理されたデータを保存する際に利用でき、「Ivory」でも採用している。

利用するには「CSVLine」オブジェクトを生成し、メソッド「addItem」の引数にデータを設定する。メソッドの実行を必要回数繰り返した後、メソッド「getLine」を実行する。このときの戻り値は CSV 一行分に相当する。以上を繰り返すことで、CSV 形式のファイルを調製できる。

実装上の問題点として、システムでライブラリ「csv.jar」の機能を利用するためには、Excel 形式のファイルを一旦 CSV 形式のものに変換するという手間が生じる点がある。また Excel では数値の桁数が多いときに「1.11+E10」のようにエラー表示されることがあり、CSV 形式に変換後のファイルにもそのエラー表示が反映されることがわかった。この問題を解決するにはライブラリ「csv.jar」に代えて Excel 形式のファイルを直接読み込むことができる機能を実装する必要がある。この点を充足するものに Jakarta プロジェクト（http://jakarta.apache.org/）のライブラリ「POI」がある。この利用方法については目下検証中である。

8.2.4. 妥当性検査

8.2.4.1. 総括

システムを介してデータ入力を行う際には、空文字対策を行わなければならない。入力されたデータはアプリケーションの機能によって SQL 文へと組み込まれ、データベースへのアクセスに利用される。データ形式が空文字であると適切な SQL 文が生成されないため、これを発行してもデータベースは SQL 文の構文エラーしか返さない。そのためデータ形式が空文字であるときは、そのまま SQL 文の呼び出しへと処理が遷移しないように、アプリケーション側にて対策を講じることが必要となる。

また、入力されたデータの形式が問題となる場合もある。例えば、メールアドレスの正確性を期するために、データ中に「@」を入力するよう促す場合が考えられる。

このように、入力情報からデータを受け取る際にはそれが妥当なデータかどうかをチェックすることが重要である（以下「妥当性チェック」という）。

これを実現するために、Jakarta プロジェクト（http://jakarta.apache.org/）にて公開されている

```
StringTokenizer st = new
StringTokenizer("Hello,World",",");
String str = st.nextToken();
```

図 9　StringTokenizer の使用例

「Validator」パッケージに注目した（commons-validator-1.1.3.jar ）。このパッケージは、XML ファイルに定義された妥当性チェックのルールを JavaBean に対して行うものである。

XML ファイルを利用するため、すでに定義づけされたチェック項目をその対象から外すことで、データ形式に対する妥当性の要件を緩和することが容易に執り行えるという利点がある。

8.2.4.2. XML ファイル

この XML ファイルでは「form-validation」タグを最上位の要素とし、その中に「global」タグと「formset」タグが入れ子要素として配置される。

「global」タグには「validator」タグを複数配置することができ、その「name」属性によって妥当性チェックの内容が決定する。例えばその属性値に「required」を設定すると、データ入力を必須扱いとすることができる。「classname」属性と「method」属性については後述する。「methodParams」属性は「Validator」パッケージの仕様に依存するため、図のとおりに記述する。最後の「msg」属性は特に設定せずとも、妥当性チェックの機能上問題ないものと思われる。

「formset」タグには妥当性チェックの対象となる JavaBean を定義するために「form」タグを配置する。このタグはチェックされる JavaBean が増えると追加される。「name」属性は JavaBean に名前をつけるもので、任意のものを設定できる。この属性値は妥当性チェックを行う際に生成される「Validator」オブジェクトの性格を基礎付けるものとして、対応する JavaBean とともに「Validator」オブジェクトのコンストラクタ引数として用いられる。

「form」タグには複数の「field」タグを配置される。ここでは javaBean のフィールド名を「property」属性に設定し、「depends」属性でフィールドに対して行う妥当性チェックの種類を設定する。例えば、物の値段を表す「price」フィールドが数値であることを保証したいときは、「depends」属性に「int」を設定する。なお「depends」属性は、「global」タグに配置された「Validator」タグの「name」属性と同一のものである必要がある。

8.2.4.3. ソースコード

本パッケージを利用するには、上記 XML ファイルの他に、妥当性チェックの結果を出力するためのメソッド群を記述したソースコードを準備しなければならない。これはメソッドの戻り値を boolean 型にしておくことで、true か false かの二値によってデータの妥当性を判定するための材料を提供するものである。その記述内容は汎用的なものであって、システムの構成等に依存するものではない。例えばデータが数値であることを保証するためには、「validateInt」と名づけたメソッドを準備して「Validator」パッケージの仕様どおりに実装すればよい。なお先に説明した XML ファイルの

```
public class ValidatorConfig※{
public static boolean validateRequired(
Object obj, Field fld){
String value = ValidatorUtils.getValueAsString(
obj, fld.getProperty());
return !GenericValidator.isBlankOrNull(value);
}
}  ※クラス名は任意でよい。
```

図 11　ソースコード（妥当性チェック）

「classname」属性はこのソースコードのファイル名であり、「method」属性はソースコードに記述されるメソッド名そのものである。

8.2.4.4.　利用の準備

ここでは、電話番号の入力を促すために、JavaBean のフィールド「telephoneNumber」を「入力必須」扱いとする場合を例に挙げて説明する。

まずソースコードにメソッド「validateRequired」を追加する。次に XML ファイルでメソッド「validateRequired」と JavaBean の名前を次のように定義する。

メソッド「validateRequired」…「required」

JavaBean…「obj_1」

そして JavaBean のフィールド「telephoneNumber」の設定を「入力必須」とするために、「Field」タグを次のように定義する。

「property」属性…「telephoneNumber」

「depends」属性…「required」

以上で準備は完了である。

8.2.4.5.　利用方法

妥当性チェックを実施するには、JavaBean を設定した「Validator」オブジェクトのメソッド「validate」を呼び出して、その戻り値を利用する。

まず、「Validator」オブジェクトを生成する際に、コンストラクタ引数に XML ファイルのリソース情報と XML ファイルで定義された JavaBean 名（ここでは「obj_1」を用いる）を設定する。XML ファイルのリソース情報は「ValidatorResources」オブジェクトで表現される。このオブジェクト生成の際には、コンストラクタ引数に XML ファイルを表すストリームオブジェクトを設定する。

```
Validator val = new Validator(resources, formName);
val.setParameter(Validator.BEAN_PARAM, bean);
ValidatorResults results = val.validate();
ValidatorResult result =
results.getValidatorResult("telephoneNumber");
boolean bl = result.isValid("required");
```

図 12　利用方法

次にメソッド「setParameter」を使って、妥当性チェックを実施する JavaBean を引数に設定する。この他に「Validator.BEAN_PARAM」を引数として設定するが、「Validator」パッケージの仕様であるため、そのまま用いてよい。

最後にメソッド「validate」を実行すると、妥当性チェックが実施される。その結果は戻り値として確保される。

戻り値である「ValidatorResults」オブジェクトには Map 型のフィールドが設けられ、妥当性チェックを受けたフィールド名をキーにして、妥当性チェックの結果を保持する「ValidatorResult」オブジェクトがマッピングされている。「ValidatorResult」オブジェクトを獲得するには、「ValidatorResults」オブジェクトで実装されたメソッド「getValidatorResult」をフィールド名（ここでは「telephoneNumber」を用いる）を引数として呼び出す。

「ValidatorResult」オブジェクトのメソッド「isValid」を呼び出すと、妥当性チェックの結果が boolean 型で表現される。引数に「required」を設定することで、フィールド「telephoneNumber」に入力がされているかがわかる。

8.3.　View

View では可能な限り HTML に忠実な内部構造が望まれる。これは HTML デザイナーが View を担当する場合、プログラムチックな内部構造は理解が困難であると考えたためである。

これを実現するために View では JSP という API を用いる。だが、View のロジックを全て JSP で記述したとしてもどうしてもプログラムチックな部分は残ってしまう。

これを回避するためにViewではJSP以外にJSTLというのAPIを用いる。JSTLはJ2EEのタグライブラリを用いて画面ロジックを可能な限りタグのみで表現するものである。

8.3.1. JSTLを用いた実装

JSTLでは式言語という特殊な言語仕様を持っており、タグの中に式言語に沿った記述を行っていく。JSTLと通常のJSPとの比較を図に示す。JSTLを用いると今までプログラムチックだった部分が大幅にカットされ、タグのみで画面ロジックを記述できている。

8.4. Controller

Controllerを実現するためにJ2EEのServletというAPIを用いる。Servletはサーバーサイドアプリケーションを実現するためのAPIである。

8.4.1. 編集処理

データベース内の情報を確認なしに変更するのはリスクが大きいため、情報の変更を行う際には必ず確認画面を表示することにした。

情報の変更は大まかに3つの手順に分けられる。

[1]編集画面を表示する。(付録2-1：編集画面を表示する)

[2]確認画面を表示する。(付録2-2：確認画面を表示する)

[3]実際に編集処理を行う。(付録2-3：編集処理を行う)

確認画面を表示することで編集を行う際の人為的間違えを減らすことができると考えた。

8.4.2. Servletのデータベースアクセス

DAOでは汎用性を高めるような設計を行っている。これはDBMSを変更しやすくするという考えの元に設計されている。

ここではDAOの利用法について、付録の2-4「登録者一覧画面の表示」を例に解説する。

[1]まずブラウザからRegistrantListServletにアクセスされる

[2]RegistrantListServletの中でDAOFactoryのgetFactoryを用いてDAOFactoryを継承したクラスを取得する。

[3]DAOFactoryからDAOを取得する。ここでは登録者の情報のやり取りを行うRegistrantDAOを用いる。

[4]RegistrantDAOのgetRegistrantList(int)を使って登録者の一覧を呼び出す。

[5]登録者の一覧をSessionに保存してJSPへ処理を渡す。

```
<form-validation>
<global>
<validator name="required"
classname="ValidatorConfig"
method="validateRequired"
methodParams="java.lang.Object,org.apache.co
mmons.validator.Field"
msg=""/>
</global>
<formset>
<form name="obj_1">
<field property="name" depends="required"/>
</form>
</formset>
</form-validation>
```

図 10　XMLファイル（定義部分のみ表示）

8.4.3. 法人名の修正

CSV形式のファイル内に誤った法人名があった場合、以前はデータベースエラーが起こっていた。だがデータベースエラーと表示されただけでは対処方法がわからないので法人名を手動で修正する方法を考えた。

まずCSVを読み込んだ後、誤りがあった場合は誤りの修正を促す法人名修正画面を表示する。そこで手動で利用者が誤りの修正が行った後、システムが修正済み法人名を元にデータ更新処理を行う。(付録2-6：法人修正画面を表示する)

8.5. データベース設計

8.5.1. テーブル設計

システムで必要なデータを正規化し、以下のような構成とした。

各表のが保持するデータは、以下の通りである。

Registrants	登録者
Companies	法人
Members	個人会員と法人を会員として結合するためのキー
Charges	会員に対する請求の履
CircuitHistories	登録者の回線使用履歴

8.5.2. ビュー設計

複数のストアドプロシージャ内で、重複する表の結合処理を行わずに済むよう、ビューを定義した。その結果、ストアドプロシージャ内のコードの見通しが良くなったと考える。

また、「法人」「個人会員」をまとめて「会員」として取り出せるよう、「Companies, Membersを結合する問い合わせ結果」と「Registrants, Membersを結合する問い合わせ結果」をUNION結合します。

8.5.3. ストアドプロシージャ

問い合わせを行うメインの表ごとにファイルを分けた。

ストアドプロシージャをまとめてjavaのクラスのように扱うことにより、ドキュメント作成に役立った。

8.5.4. ドキュメント

テーブル・ビュー・制約・ストアドプロシージャなど、データベースに関するすべてのドキュメントは、擬似的に作成したJavaコードのjavadocとした。

javadoc を使用することにより、ストアドプロシージャの java のメソッド風表示や、ドキュメント内に相互リンクができたため、作業の軽減に役立った。

8.5.5. スクリプトの生成

テーブル・ビュー・ストアドプロシージャの生成。テーブルの初期化を行うスクリプトを生成し。一括で実行できるようにした。

8.5.6. テスト

ストアドプロシージャは、ストアドプロシージャの実行を行う Java プログラムの実行結果を目視で確認する方式で行った。

8.5.7. 苦労した点

ストアドプロシージャの実装部分が一番苦労した。

いま振り返れば、PostgreSQL の環境構築前にコードを書き始めたことが一番大きな失敗だった。ストアドプロシージャで使用する PL/pgSQL 言語についてはまったく知識がなく、PostgreSQL のドキュメントを読みながら書いたため、環境構築ができた段階ではコードが全体の 7 割くらい完成してしまっていた。環境構築が終わった後に動かしてみると、文法エラーが大量に出てきてしまい、修正に大幅な時間がかかってしまった。特に、一度実行するまで文法エラーがあることすら分からない点や、エラーメッセージから文法エラーの原因特定で苦労した。

8.6. SQL の動的生成について

SQL 文の動的生成とは、ユーザから入力された値を検索するための SQL 文を Java のコードで動的に生成する技術である。作成にあたっては、GoF の Builder パターンや Hibernate の記事を参考にした。基本思想としては、SELECT 文の各要素を表すクラスを定義し、そのクラスに対してメソッドを呼び出すと、最終的に「SELECT * FROM table1 WHERE col1 = ? AND col2 <= ?」というような文字列と、「50, 100」のように SQL 文中の「?」で表されるパラメタのリストを保持するオブジェクトを生成する仕組みである。

主なクラス

QueryBuilder 格「句」統合するクラス
SelectColumns 取得する列リストを表すクラス
From 「FROM 句」を表すクラス
Where 「WHERE 句」を表すクラス
GroupBy 「GROUP BY 句」を表すクラス
Having 「HAVING 句」を表すクラス
OrderBy 「ORDER BY 句」を表すクラス
Condition Where, Having で使用する、抽出条件を表すクラス
SQLStatement QueryBuilder で作成した SQL 文を表すクラス

テスト

JUnit を使用して、単体テストを行った。

脆弱性	脅威	対策
クロスサイトスクリプティング	不正タグにより Script の実行、Cookie の漏洩が発生する。	データ出力時に記号をエスケープする
SQL インジェクション	入力データに、SQL の特殊文字が入っていた場合に不正なコードが実行されてしまう	java.sql.PreparedStatement を使用する
想定外の値入力	ラジオボタン(<input type="radio">)	リストボックス(<select>)の選択肢にない値を持ったリクエストを送る。リクエスト内の値が、HTML や JSP で書いた選択肢以外になっているかチェックする。

表 1 Web アプリケーションの脆弱性

ドキュメント

Javadoc のみを使用し、特別なドキュメントは作成していない。

また、ほぼすべてのクラスについて Javadoc 中にサンプルコードを表記した。

苦労した点

設計思想自体は単純なため、あとは細かな設計を詰め、プログラミングを行うだけだった。ただ量は多かったため、テストコード・サンプルコードを書くのは大変だった。

9. セキュリティ

平成 17 年 4 月に個人情報保護法が施行される。IVORY では 5000 人の個人情報を取り扱うため、セキュリティ的な配慮も大切だと感じた。

9.1. リスク管理

開発に使われる電子データを守るためにリスクの分析、管理を行った。

開発用データのリスク管理

脅威・・・天災、人災。

リスク対象・・・CVS のソースコードや開発用ドキュメント。データベースのテストデータ。

リスク対策・・・毎日内部的にバックアップを実行し、1 週間に 1 度、その日のバックアップを別マシンへ移す。

9.2. Web アプリケーションの脆弱性の回避

Web アプリケーションには表のような脆弱性が存在する。

10. Redhat Linux によるサーバー管理

2004 年 8 月 2 日に従来使っていたサーバーを全て Linux

絶対パス	フォルダ概要
/var/lib/pgsql/data	データベース本体が入った部分。データベースの設定もここにある。
/var/lib/pgsql/backup	データベースのバックアップを入れるところ。
/var/lib/pgsql/designDatabase	データベースの初期化のための SQL ファイル。

表 2　データベースのディレクトリ構造

へ移行した。今までは田上の開発用マシンに Cygwin を入れて擬似 Unix 環境を構築し、サーバーの運用を行っていた。Linux の導入を景気に管理の負担も軽減されるだろうと考えられる。

10.1.　採用理由

2004 年 8 月 2 日時点での Redhat の最新ディストリビューションは Fedora core 2 だった。Redhat 系を選んだ理由は、世界でもっとも有名なディストリビューションであり Web にもドキュメントが豊富にあるということからだった。だが、Fedora core 2 をインストールすると NIC が動かなくなってしまい、NIC を交換しても動かなかった。そこで NIC の稼動実績のある、Redhat9.0 をインストールすることに決めた。Redhat9.0 は NIC の拒絶反応もなく正常に動き、今日まで動作している。

10.2.　CVS サーバー

チーム間でソースコードやプロジェクト規約などを共有するために CVS を用いた。CVS はバージョン管理を行うサーバーでチーム間のバージョン管理を容易にする。

CVS でソースコードを共同所有することでデバッグによる差し替えやビルドが容易になり作業量が大幅に減った。

10.3.　データベースサーバー

Redhat9.0 では PostgreSQL は最初からインストールされている。/var/lib/pgsql に PostgreSQL がインストールされている。

/var/lib/pgsql 以下のディレクトリ構造は表のとおりである。

10.4.　ファイル共有サーバー

CVS サーバーでのデータ共有はテキストファイルに向いているが、一時的なファイルの共有やファイルサイズが大きなファイルの共有には向かない。そのためファイルサーバーを構築し、ファイル共有を実現したい。

今回のファイル共有サーバーは Samba サーバーを用いた。Samba は WINS による名前解決ができるため、学内ネットワークとホスト名の同期が取れると考えた。

11.　検証

検証はソフトウェアテストとメトリクス計測によって行う。

11.1.　ソフトウェアテスト

テストは Junit を用いた単体テストを行う。テストが可能な Model は Junit による単体テストを行い、テストを全てパスした後は Cactus による Servlet のテストを行う。

11.1.1.　Model

テストデータを「JUnit」で設定して、妥当性チェックの動作を標準出力で確認できるようにした。本システムでは妥当性チェックが機能して警告を発する場合には、制御が例外処理へと移るように実装している。そのため例外処理に移ったかどうかで出力内容を変化させるようにテストコードを作成した。

データベースアクセスの役割を担う「DataAccessObject」の単体テストは、現時点では実現できていない。これは「DataAccessObject」の動作に「DataSource」を利用したため、サーブレットによる呼び出しが必要となったからである。そのため「JUnit」を使用することができず、結合テスト時に目視による確認をするに留まっている。

「DataAccessObject」そのものの設計について見直しを考えると同時に、その単体テストに手法について検討中である。

11.1.2.　View

View は機械的にテストするのが難しく、目視によるテストを行うことにした。View のテストは HTML で作成した画面レイアウトと、実際に動作している JSP と比較して配置などが正しいかどうかを検証する。

11.1.3.　Controller

Servlet のテストには cactus という jakarta API を用いる。cactus は Servlet、JSP、EJB などの Web アプリケーションのテストを行うための API である。

Controller でテストすべき大まかな項目は以下のとおりである。

(1)全ての Servlet が望むべき機能を果たしているか。

(2)全ての Filter が望むべき機能を果たしているか。

(3)全ての JNDI が名前解決を行えているか。

実際に cactus を使ってみて、(2)と(3)はテストできない。(2)では IVORY の Filter 機能をテストには cactus はテストのための機能が少なすぎてテストできない。(3)では cactus は JNDI の名前解決を行う機能をサポートしていない。

(1)については、画面で入力されるであろう値を Servlet に渡して実行してみることでテストを行った。今回は全ての Servlet で、例外が飛ばずに動作するかをテストした。

(2)では、編集処理の Servlet が Filter をきちんと通っているか実際に編集処理の Servlet を通して目視でテストした。

(3)では、Servlet を通じてデータベースへアクセスを行いデータがきちんと取れているか目視でテストした。

11.2.　メトリクス計測法によるソフトウェア品質の検証

IVORY のソフトウェア品質を計測するためにメトリクス計測法を使った。メトリクス計測法では、主に以下のような指標を用いてソフトウェアの品質を計測する。

・サイクロマティック複雑度

一つのメソッドにおける if 文の数。少ないほどソースを理解しやすい。

・ソフトウェアのソースコード行数

コメント行や空行、カッコ行を含まない純粋な Java ソースコードの行数。多いほど大きなソフトウェア。

・ネストの深さ

if 文や for 文などがどれほど深い階層になっているか。少ないほどソースを理解しやすい。

・クラス内におけるメソッドの数

少ないほどモジュール分割がされている。

上記の代表的な指標を用いてソフトウェアの品質の検証を行った。計測数値を表に示す。

11.3. データベース

データベースの検証は、データベースの操作を行う Java プログラムの実行結果を目視で確認する形をとった。本来であれば一定のルールに基づくテストデータを用いた自動テストを行うべきところだが、時間的制約から見送った。

テスト用 Java プログラムは、データベースの変更を行ったトランザクションをロールバックすることにより、同じテストを繰り返し実行できるようにした。

	合計	中央値	標準偏差	最大値
サイクロマティック複雑度		1.505	1.298	21
ソースコード行数	8322	6.941	10.652	99
ネストの深さ		1.296	0.64	8
クラス内メソッドの数	1181	4.631	5.935	47

表 3 メトリクス計測

12. 納品・試験運用

2004 年 12 月 22 日第 1 回目の納品を行った。納品時は主に説明書を見ながら、機能の説明書を行う予定だった。

本番環境は図-納品時の環境のとおりであった。

だが、いざ操作説明を始めようとした途端、データベースにアクセスできなくなるという問題が発生し、システムが動かなくなってしまった。仕方なくシステムなしで納品を進めた。問題は納品中に解決したが、とりあえず今回はシステムを持ち帰り翌年 2 月 4 日に再挑戦を行うことになった。

2005 年 2 月 4 日に第 2 回納品を行った。第 2 回納品は前回の失敗の要因の一つだった環境を見直して、Web アプリケーションが本来実装されるネットワーク環境へ実装をした。利用者はサーバーへアクセスし、ブラウザから機能を利用するという方法を取った。このときの本番環境は図-納品時の環境のとおりである。

今回の納品は比較的スムーズに進んだが、納品したシステムと利用者が望むシステムとの食い違いがまだ大きかった。今回の納品は仮運用ということで、使っていただけることに

1回目

CPU	AMD 450MHz
メモリ	64MB
HDD	5.7GB
OS	Windows98 SE

2回目

CPU	Intel Pentium2
メモリ	128MB
HDD	6.0GB
OS	Redhat Linux 9.0

図 16 納品時の環境

なった。

13. 今後の課題

研究を行っている最中に考え付いた課題を列挙していく。

13.1. IVORY の開発の課題

13.1.1. XP 手法

今回の研究では CVS サーバーによるソースコードの共同所有という XP 手法を採用して開発を行った。ソースの共同所有はチーム間のバージョン管理の手間を省くという点で一定の効果があったと考えている。今後も XP によるシステム開発を推進していきたいと考えている。今後システム開発において導入効果が期待されるであろう XP のプラクティスを紹介する。

・テストファースト

通常のウォーターフォールモデルでは実装の後に単体テスト、結合テストという流れである。だが、テストファーストでは単体テストのテストケースを作ってから実装を行い、結合テストを行う。

・継続的インテグレーション

開発中も常にシステムをビルドしつづけ、テストに 100%通る状態を保つことを継続的インテグレーションという。このプラクティスは常にテストが完了した状態を保つことで信頼性が向上するという効果がある。従来のテストでは開発途上でモジュールが編集されると設計、実装段階まで作業が手戻りするなど非常な手間が発生していた。手戻りが発生する

とテストケースを再構築し、テストを実行しなければならなかった。だがXPでは変化に強いシステムを作ることが目的であるため、継続的インテグレーションを導入することで変更があっても即座にテストを行い、システム全体の因果関係によって発生するバグを検知することができると考えられる。

次回の実装では上記の二つのプラクティスについて検討していきたい。

13.1.2. IVORY新機能

13.1.2.1. Microsoft Excelから直接データの読み書き

今回のシステムではカンマ区切りのCSV形式のファイルに変換してから読み書きを行っていた。だが、これではシステムへ投入するたびに拡張子を変えなければならないため手間がかかってしまう。これを回避するために拡張子を変えずにExcel形式で直に読み書きができる機能の実装をしたい。

これを実現するためにJakarta POIというライブラリを用いる。これはExcelを読み書きするAPIで、これを使えば拡張子を変えずにシステムへのデータ投入が可能になる。

13.1.3. データベースアクセス

13.1.3.1. JNDIによるオブジェクト管理のコスト

JNDIはデータソースなどのオブジェクトをTomcatコンテナが管理する機能であるが、これを利用するとシステムリソースをたくさん消費することがわかった。

今回のシステムではDAOを呼び出すたびにJNDIを呼び出しているため今後は接続を使いまわせるようなシステムの構築を目指す。

13.1.4. IVORY開発におけるネットワーク環境

13.1.4.1. Linux環境

2005年になってみてディストリビューションをもう1度見直してみると、今回の研究に最適だったのはSuSE Linuxだったのではないかと感じる。SuSEはNovell社が開発しているRedhat系ディストリビューションでJavaの開発や、Tomcatなどもサポートしている。今後はSuSEへ乗り換えたいと考えている。

SUSE導入のメリット

・Tomcatを始めとした、Javaによる Enterprice開発をサポート。

・MandrakeLinux、Fodoraに次ぐメジャーディストリビューションであり、パッケージ管理にはRPMをサポートしているため比較的移行しやすい。

・これまで自前が用意していたShellがほとんど実装済み。

SUSE導入のデメリット

・最新版の9.2はネットワークインストールとDVDイメージからのインストールしかできない。

・IVORYの稼動実績が現状では証明されていない。

・開発環境と本番環境でディストリビューションが異なる。

13.1.4.2. Subversionによるバージョン管理への移行

開発ではCVSを用いてバージョン管理を行っている。だが現行の最新バージョン管理パッケージはSubversionであり、CVSの欠点を補完したものとなっている。CVSの欠点は以下のようなものがある。

・ディレクトリは管理されない

・ファイル名が変更された場合、サーバー上で変更前のファイルは消えてしまう

Subversionでは上の問題は回避されている。だがCVSによるバージョン管理は充分使えるレベルで機能しており、この課題の優先度は低い。

13.1.5. パフォーマンス

今回のシステムでは4000件のデータを扱うため、ストアドプロシージャを利用してデータベースアクセスを早くするなどの気を使った。だが、DAOの無駄遣いや妥当性検査のコストなど、ソースコード面でまだまだ課題が山積している。

今後はデータベースのみならず、設計面でのパフォーマンス向上に努めたい。

13.2. ebonyデータベースの課題

13.2.1. システム情報のDB化

システムのエラーメッセージ、ログのデータベース化をしたい。

エラーメッセージはServletのソースコードの直書きしており、メッセージが変わる度にソースコードを変更するため危険である。エラメッセージをデータベース化することで数字でメッセージを呼び出すことができ、変更が容易になると考えられる。

ログはIVORY側では取っていない。システムを運用していく上で、利用者がどのような操作を行って不具合が発生したかなどを検証するためにログは不可欠である。だが、モジュール内にログのメッセージを書き込んでしまうと記述ミスや記述漏れが発生する可能性がある。これを回避するためにアスペクト指向をもちいてログ機能の実装を行いたい。また、それをDB化することでログの検索もやりやすくなると考える。

13.2.2. テストの見直し

テストについては時間の関係もあって、満足行くものが出来上がらなかった。今後はもっと厳密な値の検証やチェックするべきテスト項目を増やしていきたい。

13.3. フレームワークの適用

今回のシステムでは時間の関係でフレームワークを適用しなかった。だがフレームワークを適用することで得られる効果は大きく、今後もフレームワークの適用は検討していく。

今回適用したいフレームワークについて紹介する。

・Struts

言わずと知れた業界標準のMVCフレームワーク。

・Seaser2

アスペクト指向プログラミング、O/Rマッピングをサポートした国産の多機能フレームワーク。ドキュメントも全て日

本語。

・Spring

Seaser2 と競合関係にある多機能フレームワーク、Seaser2 よりも多機能で複雑なアスペクトを記述できる。

14. 環境の自動化

今回の研究では、CVS によってバージョン管理を自動化したが、これらの自動化は他にも応用できる。その実装方法を以下に示す。

14.1. テスト実行の自動化

継続的インテグレーションは常にビルドを繰り返し、モジュールの追加、変更、削除があった場合でも 100%テストにパスするような状態を保つことが目的である。

これを実現するためにビルドのためのマシンを 1 台用意する。このビルドマシンは可能な限り本番環境に近い方が理想である。ビルドマシン上では定期的に CVS からソースコードを読み取って、テストの実行を行いテスト結果をレポートとして XML に吐き出す。それを HTML に変換、Tomcat で Web 上に表示させる。

開発者は CVS へテストケースをコミットするだけでよく、テスト実行の手間を大幅に省くことができる。(図 : テスト自動化のイメージ)

この自動化のポイントは以下のようなものである。

[1]テストケースを定期実行

→UNIX のデーモン Cron でテストを定期実行。間隔はチームの誰かがコミットを 1 回行う期間が理想。

[2]テスト結果をレポートとして XML に変換

→Junit にこの機能がついている。Ant によって呼び出すことが可能。

[3]XML を HTML に変換

→XSTL という技術を使って XML を HTML に変換。

[4]Tomcat で表示

→HTML で表示。

14.2. 本番環境への更新の自動化

AES の本番環境で何らかの不具合が発生した場合、不具合を調査、修正し、AES へ出向いてシステムを入れなおす作業が発生する。

CVS を使えば AES へ出向かずに本番環境の更新を行うことができる。(図 : 本番環境更新のイメージ)

15. 引継ぎについて

来年は顧客管理システムの新規機能追加という研究テーマで 3 名の新規研究生が参加する。

現在ではオブジェクト指向開発などの基礎を固めるためゼミのような形式で特定のテーマについて発表をしあい、意見を交換し合っている。

16. 所感

田上・・・今回の研究では Linux を用いた開発、チーム開発など様々な新しいことを勉強でき、とても有意義だった。今回の研究ではほとんどのことをチーム間でやっており、チ

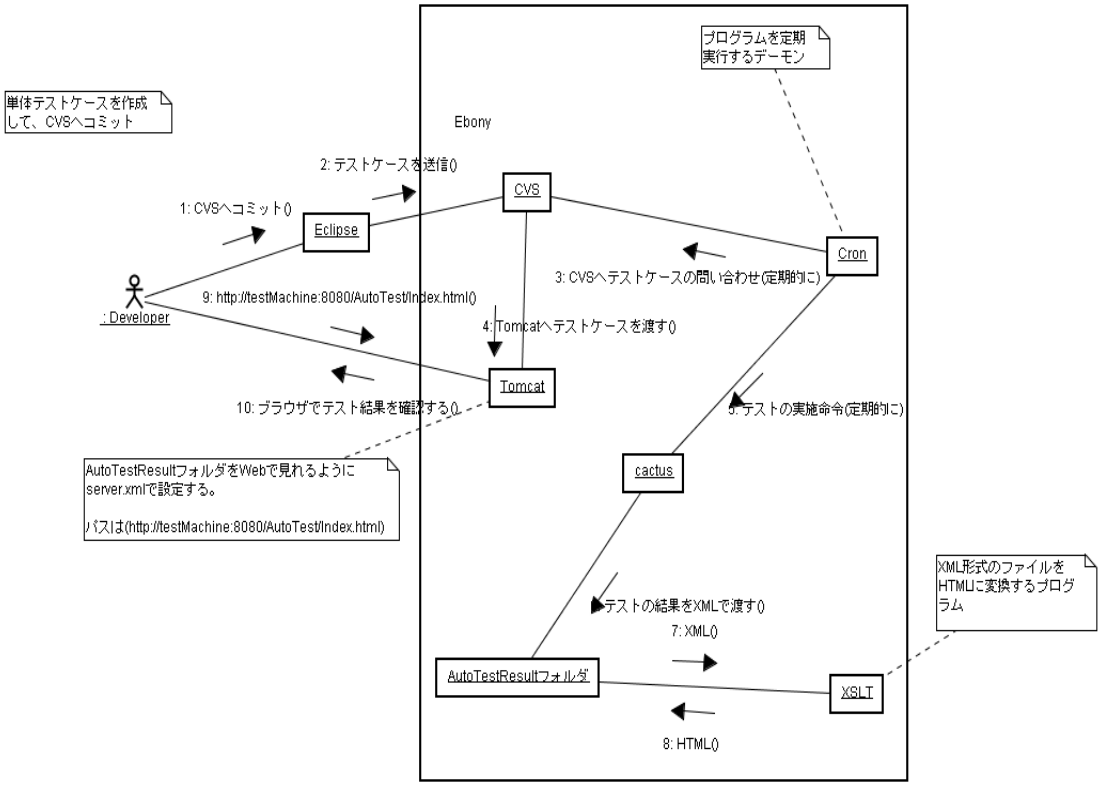


図 17 テスト自動化のイメージ

ーム内での会議なども非常に有意義なものだった。だが、少々

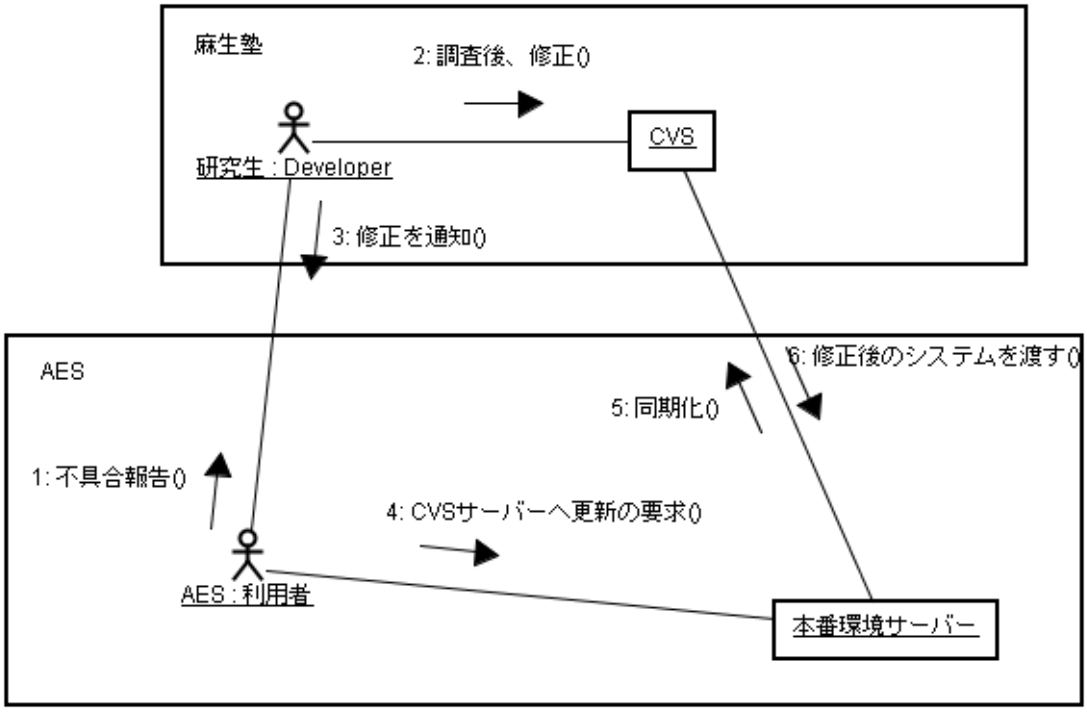


図 18　本番環境更新のイメージ

チーム内で突っ走り過ぎたところもあり、もっと先生やAESさんとコミュニケーションを取っていればよかったと思うこともしばしばあった。そして同時に、コミュニケーションの大切を学んだ1年であった。

また、チームのふたりにも非常に感謝している。

上田君は高校時代にもシステム開発の長いキャリアがあり、非常にチームに貢献してくれた。上田君のおかげでデータベースは、かなり上等なものができあがった。

服部君はSJCPを持っているとはいえ、キャリアがある二人によく1年もついてきたと感心している。服部君の横で、内部設計に関して暴走する二人は見てて不気味だったのではないかと、正直反省している。

上田・・・今回一番苦労した原因は、業務化するシステムが複雑だったことである。

必要な機能の量を考えると、3人で行うには大きすぎる規模なのではなかったと思う。また、システムに実装した機能についても最初からすべてが分かったわけではなく、話し合いのたびに次から次へと新しいことが出てくる。という状態だった。今振り返ると、2月に1度くらいのペースで打ち合わせを続けていれば、もっとよい結果になっていたのではないかと思う。

11月のあたりから、クラスの卒業研究のため作業時間が取れなくなり、田上君と服部君には本当に苦労を掛けてしまった。申し訳ないと思っている。

服部・・・研究室での活動は発見の連続だった。デザインパターンやライブラリの活用等、効率的な開発のために何をしなければならないのか、研究室に入室していなければ実務に携わるまで学ぶことはできなかったと思う。実案件を担当させていただいたことで、本を読むだけでは習得が困難な「システム開発の流れ」について体験することができた。このことは私にとって大きな糧となったと実感している。また先生方からご指導をいただく中、田上君・上田君には技術面での後押しを受け感謝せずにはいられない。こうした中で「システムの納品」という目標を達成することができて大変満足している。この一年で得た貴重な経験を下地にエンジニアとして精進していきたい。

謝辞

今回の研究の立ち上げに尽力してくださったIT教育推進室の徳重稔先生、横山新先生に深く感謝いたします。

また、今回の研究の顧客として協力していただいた、麻生教育サービス(株)の磯口晴美様に深く感謝いたします。

参考文献

[1] 米山 学　SUN教科書　Web Component Developer(試験番号310-080)翔泳社

[2] 原 一郎(監修)　Sun Certified Programmer for the Java2 Exam(310-035)

[3] ソフトバンクパブリッシング 三島 俊司(監修) サーブレット/JSP プログラミングテクニック 改訂版

[4] ソフトバンクパブリッシング 結城 浩　増補改訂版

Java 言語で学ぶデザインパターン入門

[5] ソフトバンクパブリッシング 結城 浩　Java 言語で学ぶデザインパターン入門 マルチスレッド編

[6] ソフトバンクパブリッシング 林 晴比古 新 Linux/UNIX 入門 ソフトバンクパブリッシング

[7] 月刊 JavaWorld　2004 年 5 月号～11 月号 IDG ジャパン

[8] atmarkIT (http://www.atmarkit.co.jp/)

[9] Java2SDK,Standard Edition ドキュメント v1.4.0 (http://java.sun.com/j2se/1.4/ja/docs/ja/index.html)

[10] Java2 Platform, Enterprise Edition, v1.3.1 API 仕様 (http://java.sun.com/j2ee/sdk_1.3/ja/techdocs/api/index.html)

[11] 日本 PostgreSQL ユーザ会 (http://www.postgresql.jp/)

[12] The Ja-Jakarta Project (http://www.jajakarta.org/)

[13] CSV(Comma Separated Value)を扱うクラス CSVTokenizer & CSVLine (http://www.wakhok.ac.jp/~tomoharu/Java/csv/)

[14] Junit.org (http://www.junit.org/index.htm)

付録

付録 1.IVORY 取扱説明書 version1.1

付録 2-1.編集画面を表示する

付録 2-2.確認画面を表示する

付録 2-3.編集処理を行う

付録 2-4.登録者一覧画面を表示する

付録 3-1.ebony E-R 図

付録 3-2.会員ビュー

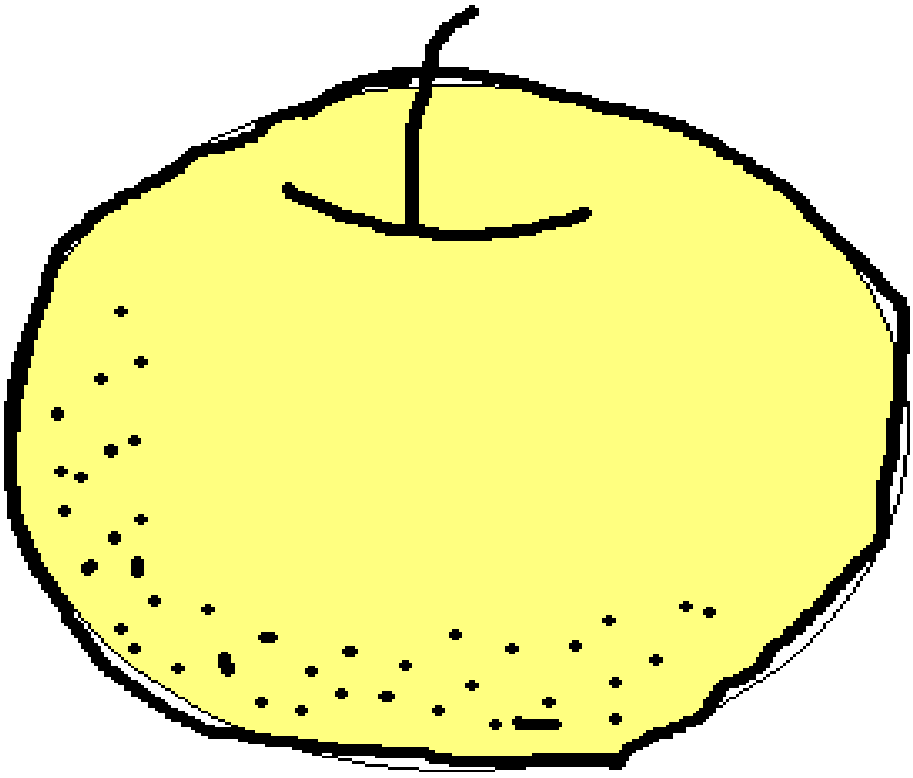
付録 4.SQL の自動生成クラス図

付録 5.MVC のイメージ

2004/12/15
version 1.0

麻生塾
IT教育推進室

IVORY取扱説明書



プロジェクトチーム「清流」

目　　次

0　取扱説明書について
1　Ivory について
2　会員管理機能
　(ア) 法人情報の閲覧
　(イ) 法人に所属する登録者の表示
　　①　登録者の検索
　　②　登録者の並べ替え
　　③　登録者情報の CSV ダウンロード
　　④　任意の登録者の回線変更履歴の表示
　　⑤　任意の登録者の変更
　(ウ) CSV を用いたデータ更新
　　①　登録者の追加
　　②　登録者の変更
　　③　登録者の削除
　(エ) 法人情報の変更
3　会計管理機能
　(ア) 法人の会計情報の表示
　　①　月別累計データの表示
　(イ) 個人の会計情報の表示
　(ウ) 請求情報の更新
4　故障かなと思ったら
　(ア) 症状と原因、そして解決策
5　その他
　(ア) 用語解説
　(イ) 法人会員の追加と削除
　(ウ) バグの報告

0. 取扱説明書について

本書は Ivory を扱う上での注意や操作方法を記述したものです。
操作を始める前にパソコンの電源がきちんと入っているかを確認してください。
また画面に表示されている住所などの情報はテストのために作成されたもので架空のものです。

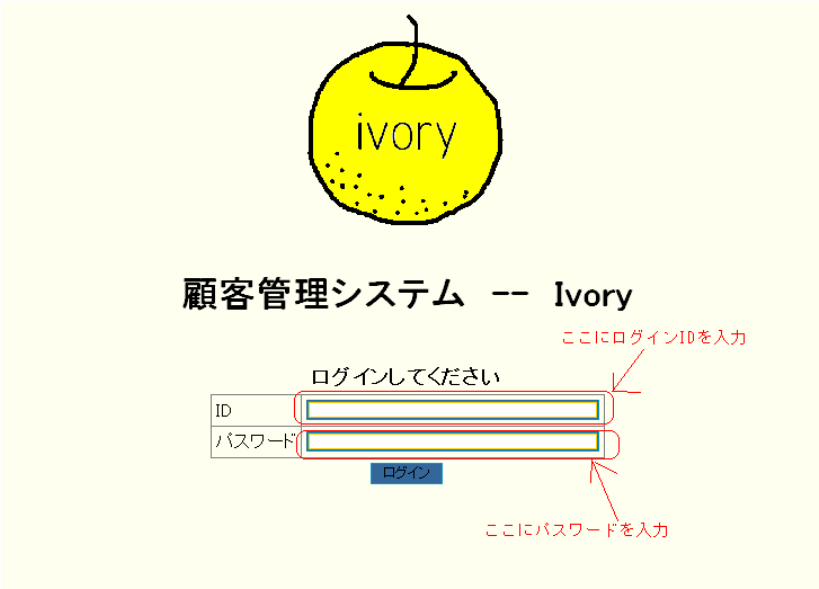
1. Ivory について

Ivory は Web 上で動作するアプリケーションです。
本アプリケーションの機能は
会員管理機能：会員を閲覧、編集を行う機能
会計管理機能：請求処理の進捗を確認・記録を行う
と大まかに分けて二つの機能で構成されています。

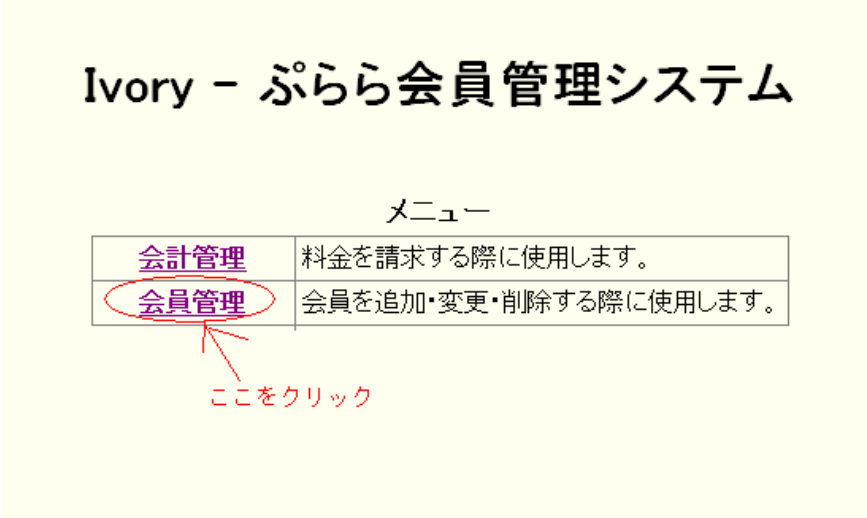
2. 会員管理機能

この機能は会員管理画面を起点として実行されます。
では会員管理画面の表示までの操作を行います。

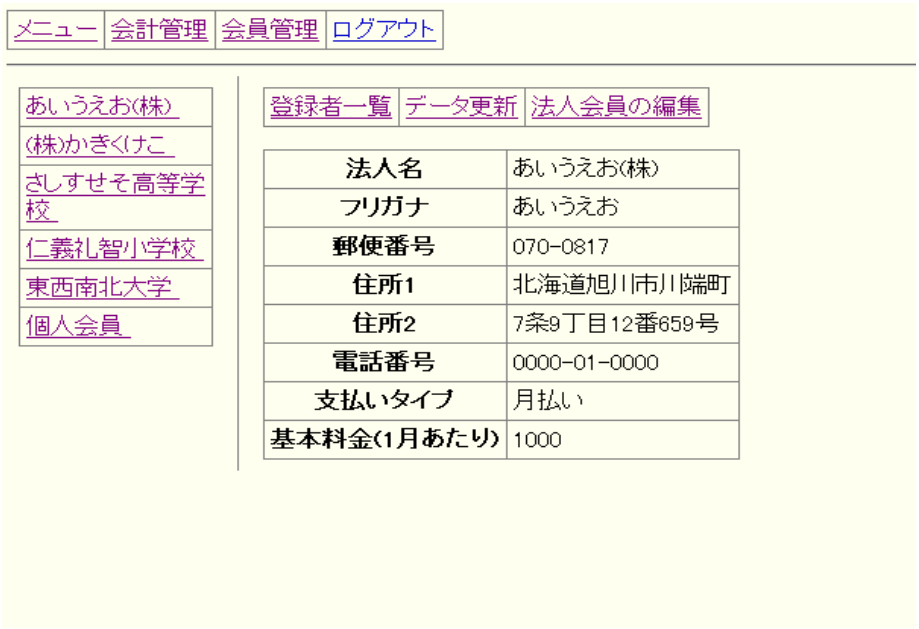
[1] ログインを行います。Ivory へアクセスすると、まずログイン画面が表示されるので、[ID]欄に Ivory の利用者 ID を、[パスワード]欄にパスワードを入力してください。



[2] メニュー画面が表示されました。「会員管理」をクリックします。

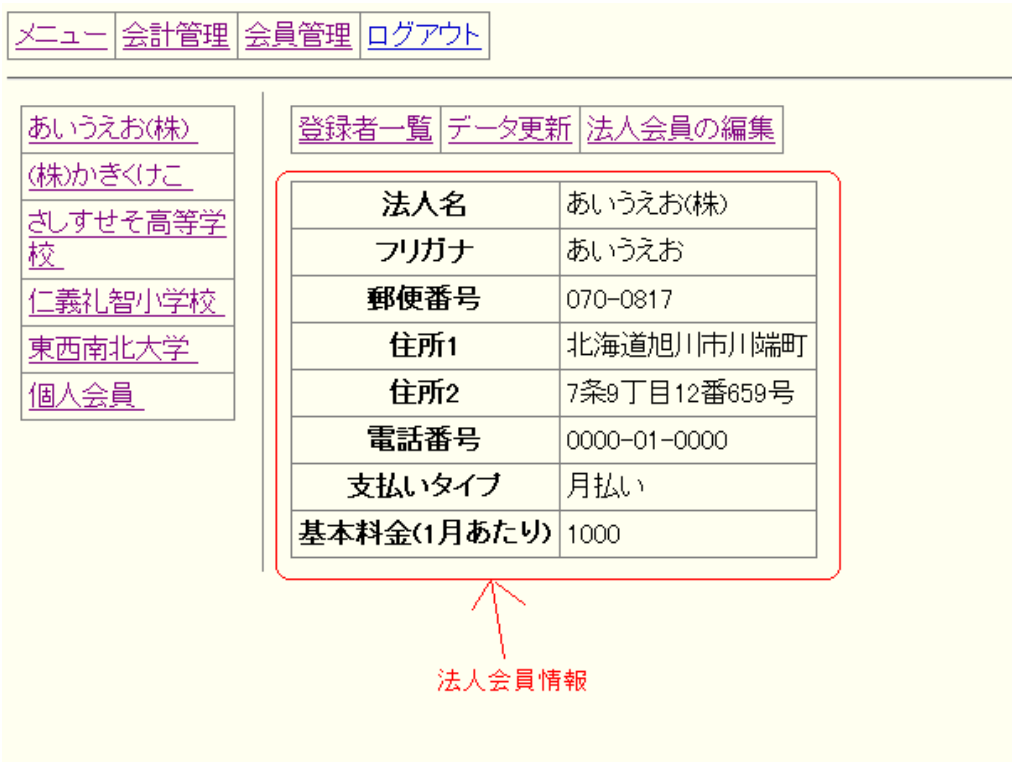


[3]　会員管理画面が表示されました。



(ア) 法人会員の閲覧

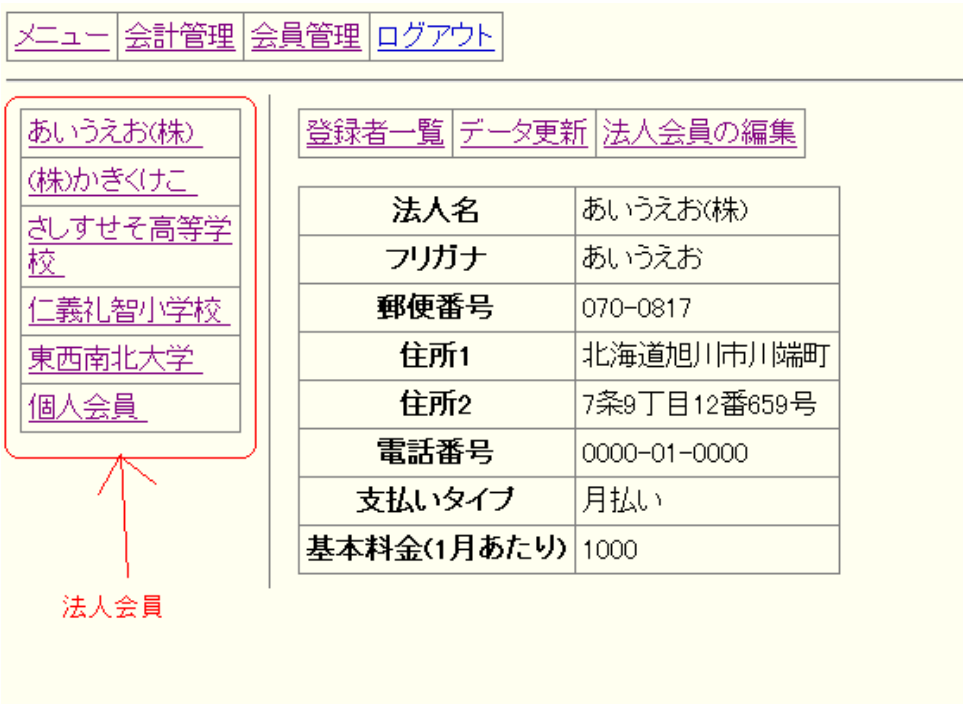
会員管理画面では法人会員の情報を表示します。



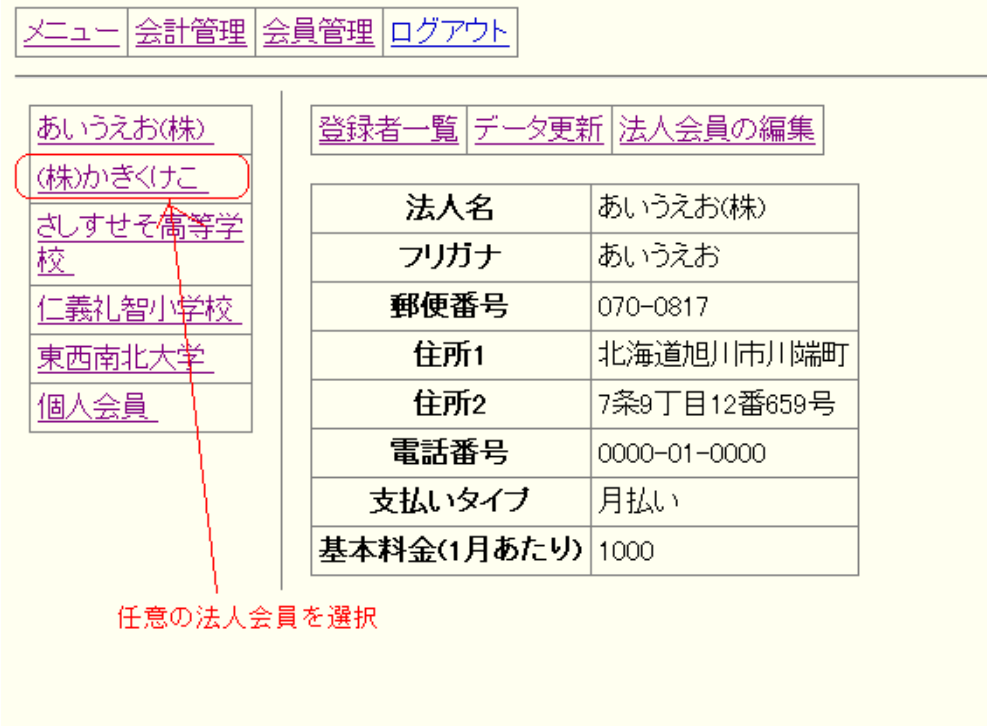
1. 他の法人会員情報の表示

   他の法人会員の情報を見たい場合は、左側の法人会員のリストの中から選択します。

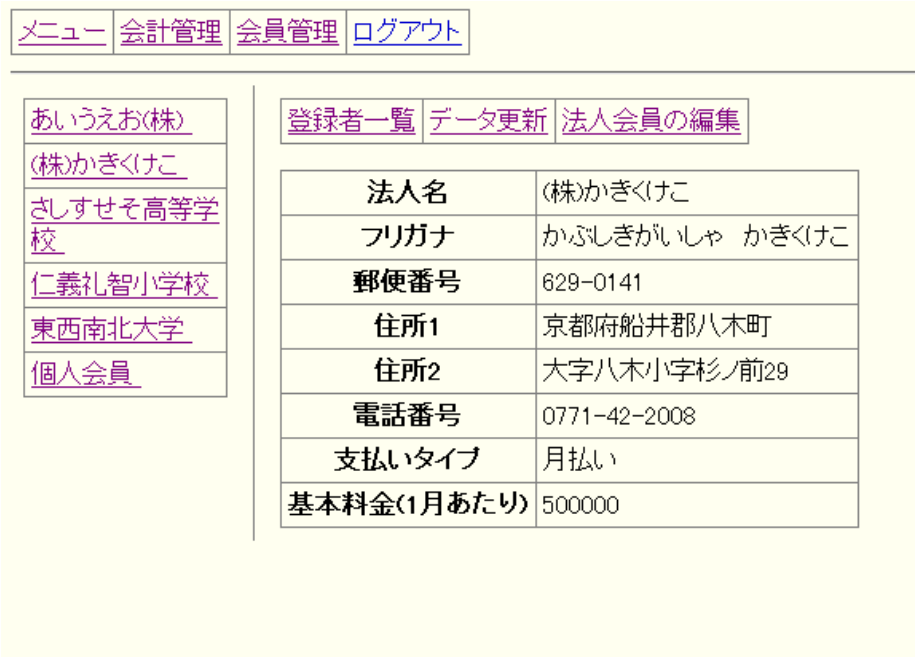
   [1] 会員管理画面の左側に法人が複数あります。



   [2] 任意の法人会員のリンクをクリックします。



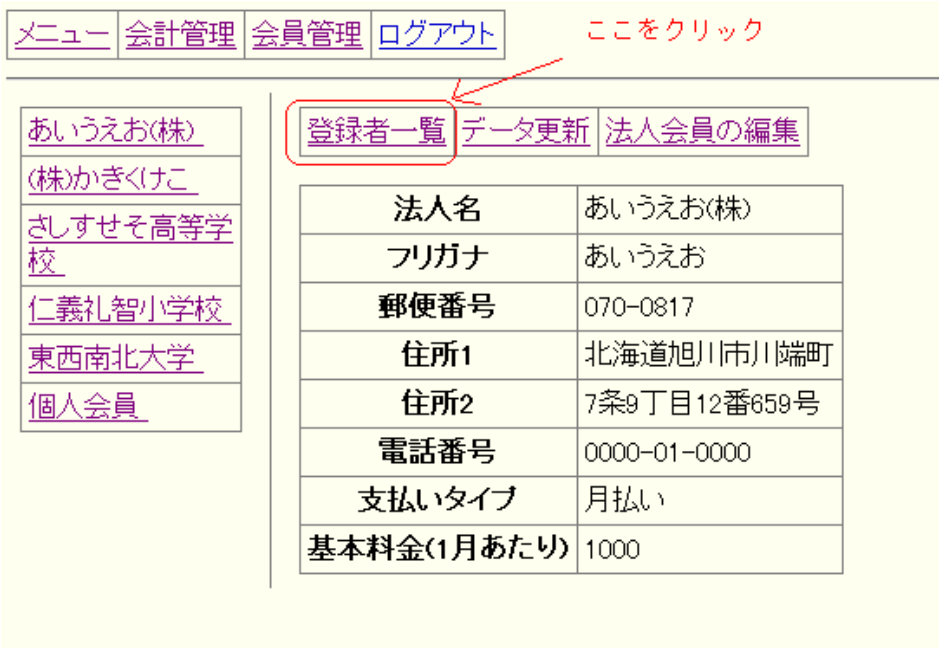
[3]　画面に任意の法人会員の情報が表示されました。



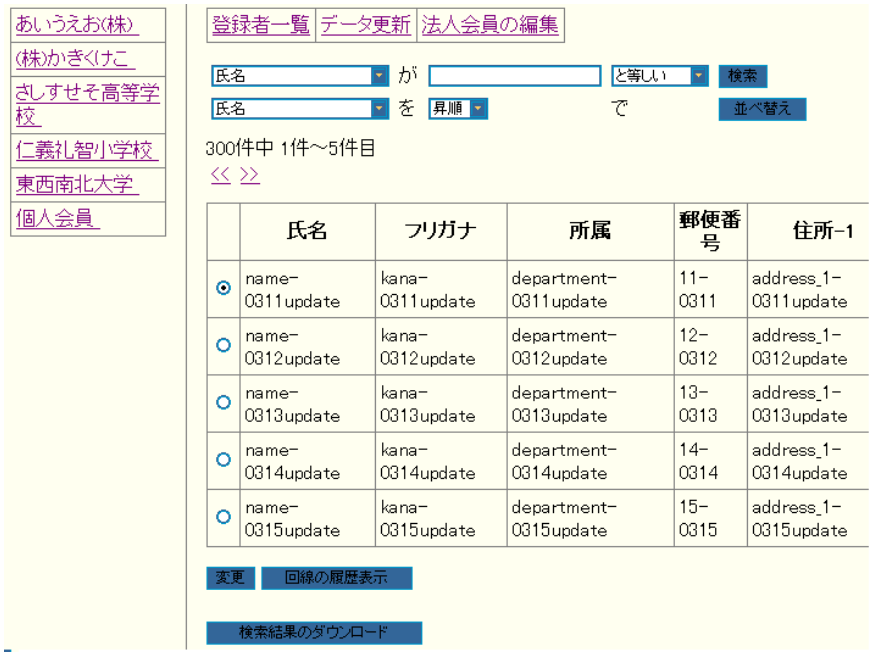
（イ） 法人会員に所属する登録者の表示

現在選択中の法人会員に所属する登録者の表示を行います。

[1]　会員管理画面で、「登録者一覧」のリンクをクリックします。



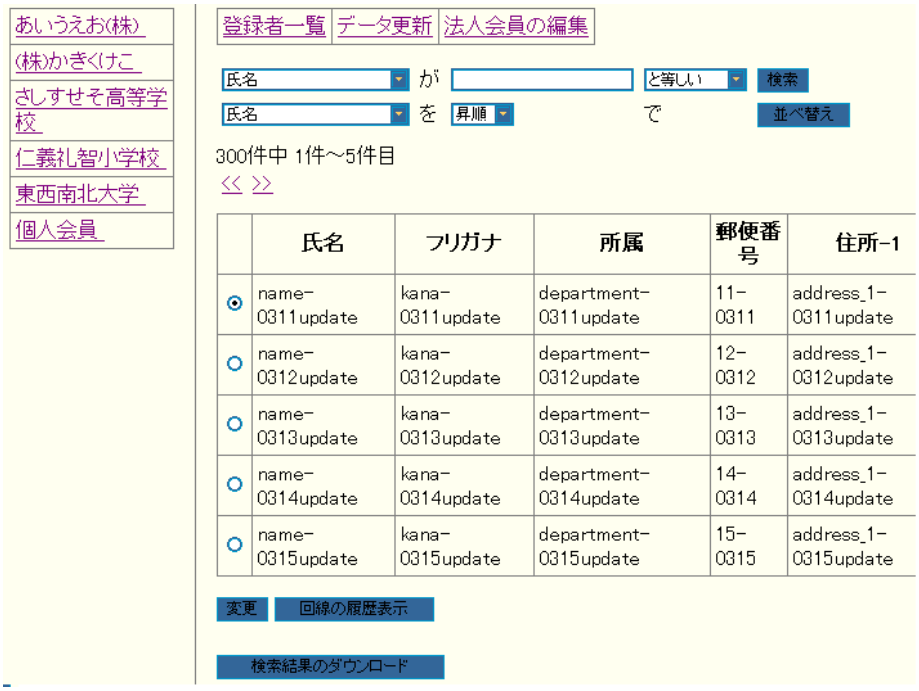
[2]　現在選択中の法人会員に所属する、登録者の一覧が表示されました。



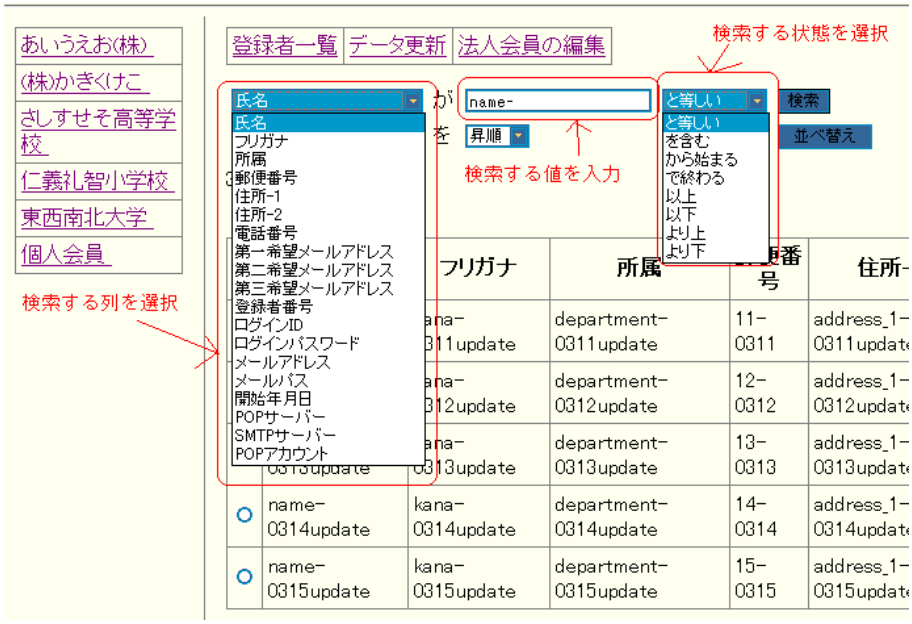
1.　登録者の検索

現在表示されている法人会員の登録者一覧から特定の登録者を検索することができます。検索されるのは現在選択中の法人会員に所属する登録者のみです。また、検索結果、並べ替え結果をさらに検索する、「絞込み検索」も可能です。

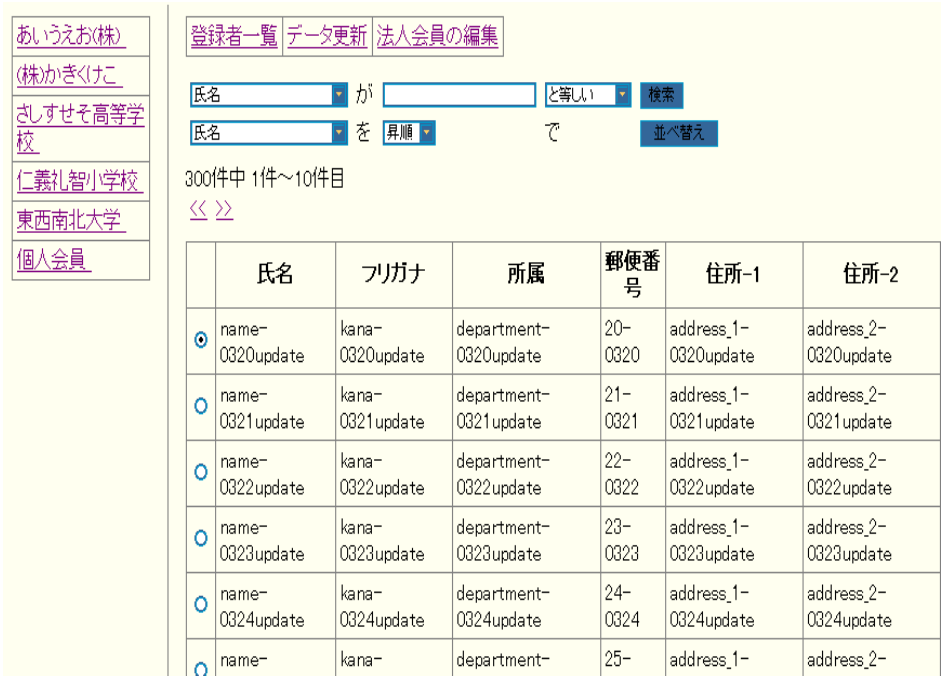
[1]　登録者一覧画面を表示します。



[2] 検索の欄に検索する列名、検索する値、検索する状態を入力します。その後、「検索」ボタンを押します。ここでは氏名が「name-032」で始まる登録者を検索します。



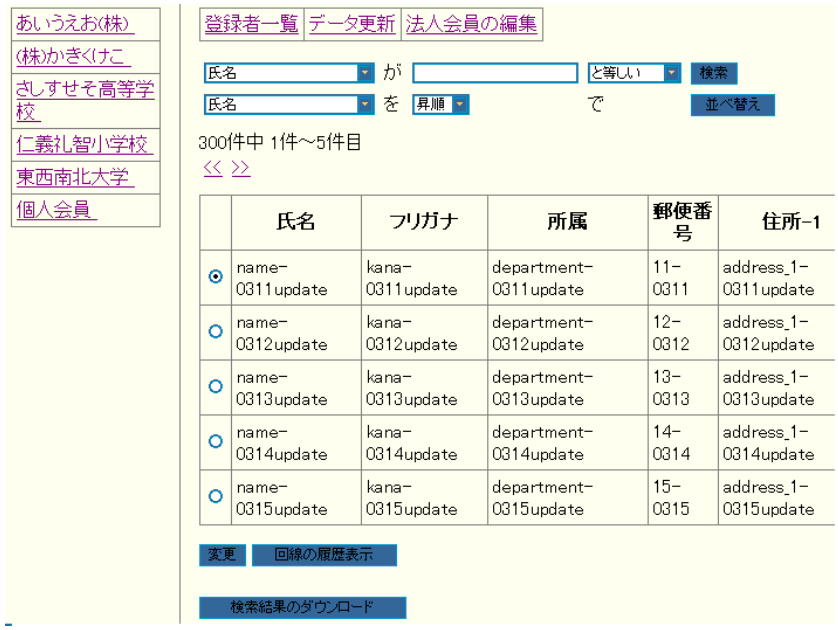
[3] 検索結果が表示されました。



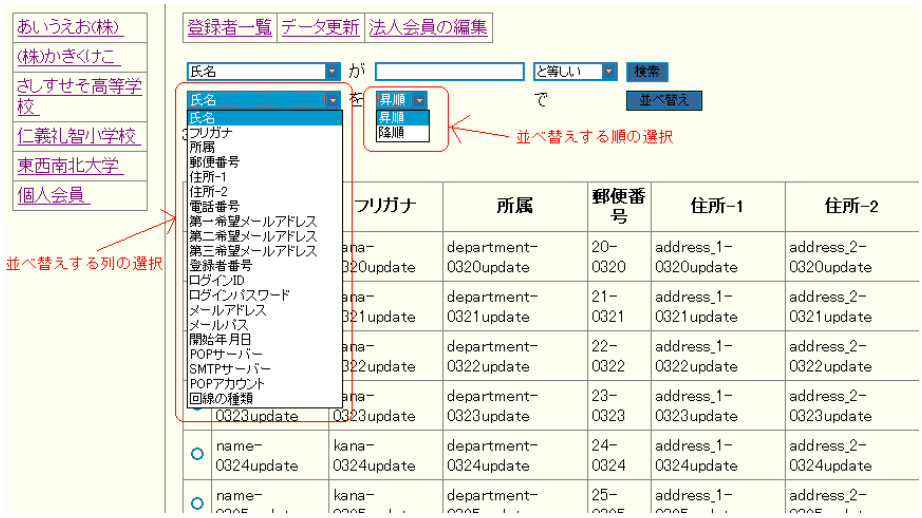
2. 登録者の並べ替え

現在表示されている法人会員の登録者一覧から全ての登録者を並べ替えすることができます。並べ替えされるのは現在選択中の法人会員に所属する登録者のみです。また、並べ替え結果、検索結果をさらに並べ替えする、「絞込み並べ替え」も可能です。

[1] 登録者の一覧を表示します。



[2] 並べ替えする列、並べ替えする順を選択します。ここでは「名前」を「降順」に並べ替えしてみます。

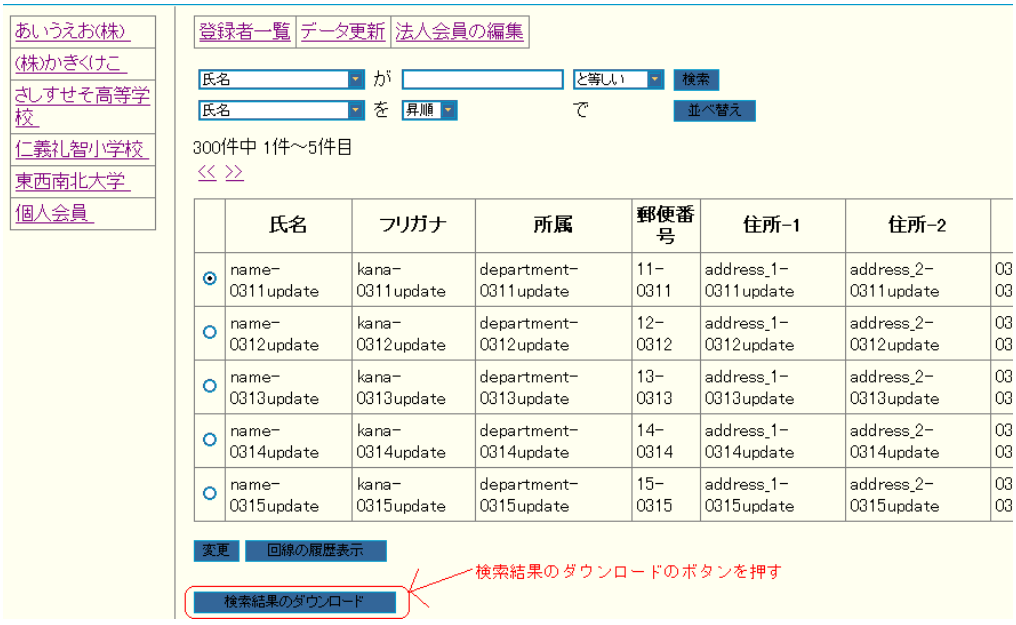


[3]　並べ替え結果が表示されました。

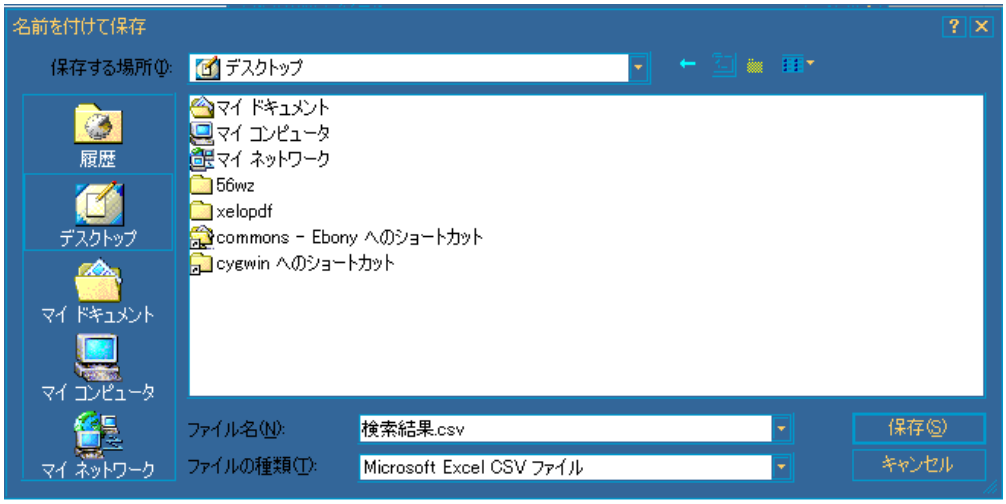
3.　登録者情報の CSV ダウンロード

現在、登録者一覧画面で表示されている登録者の一覧を CSV でダウンロードすることができます。また、検索を行った後の検索結果もダウンロードすることもできます。

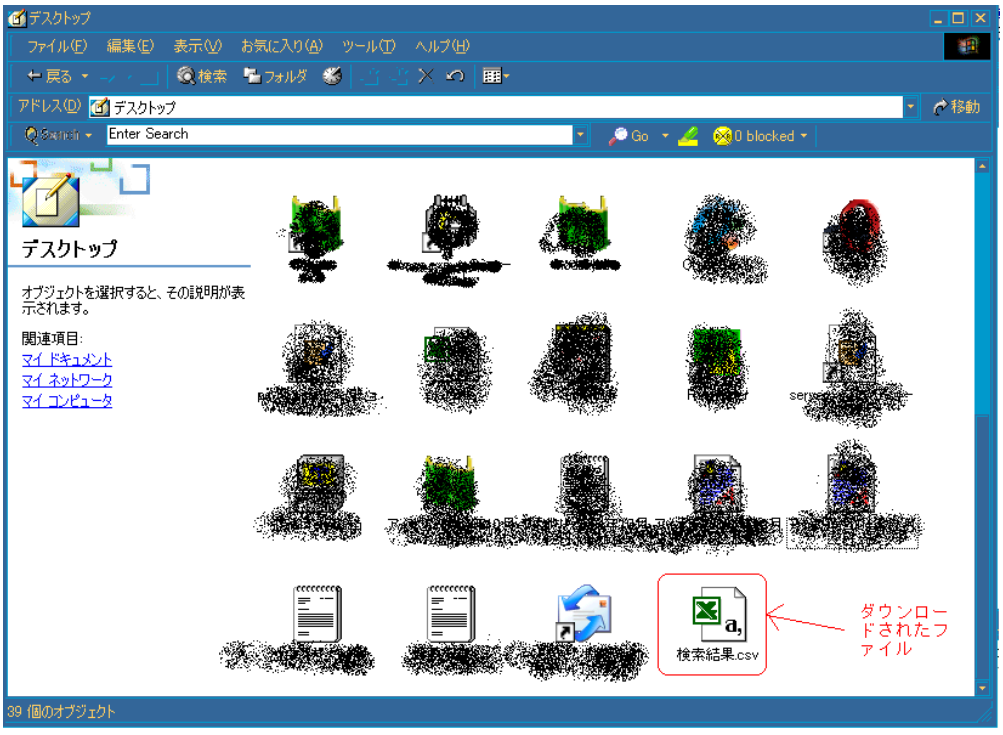
[1]　登録者一覧画面で「検索結果のダウンロード」ボタンを押します。



[2]　任意の場所に保存します。



[3]　CSV 形式で保存されました。



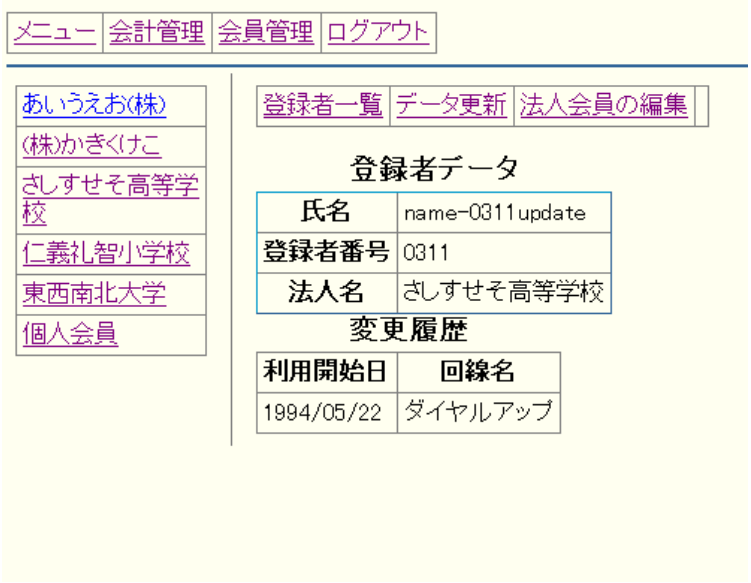
4. 任意の登録者の回線変更履歴の閲覧

任意の登録者の回線変更履歴を表示させることができます。

[1] 登録者一覧画面で任意の登録者を選択します。

[2] 「回線の履歴表示」ボタンを押します。

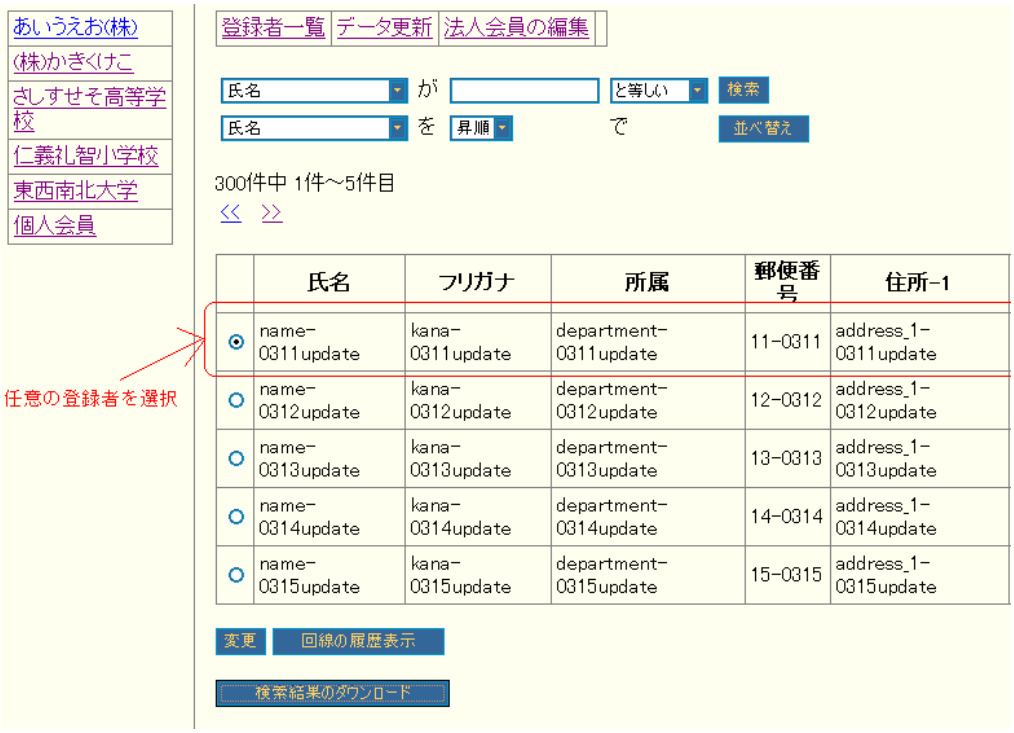
[3]　任意の登録者の回線の変更履歴が表示されました。



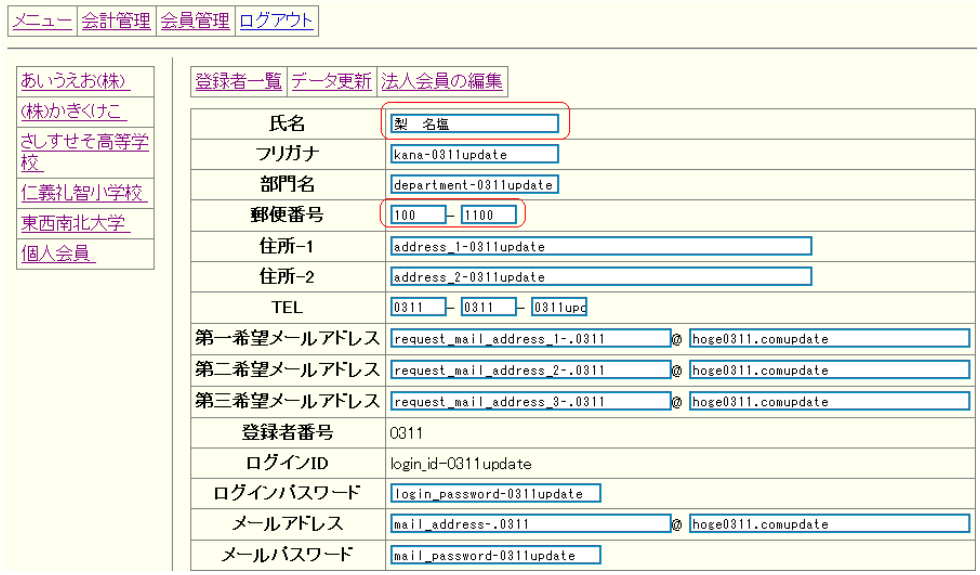
5.　任意の登録者の変更

登録者の情報を変更します。回線の上昇変更を行った場合、請求情報が自動で追加されます。

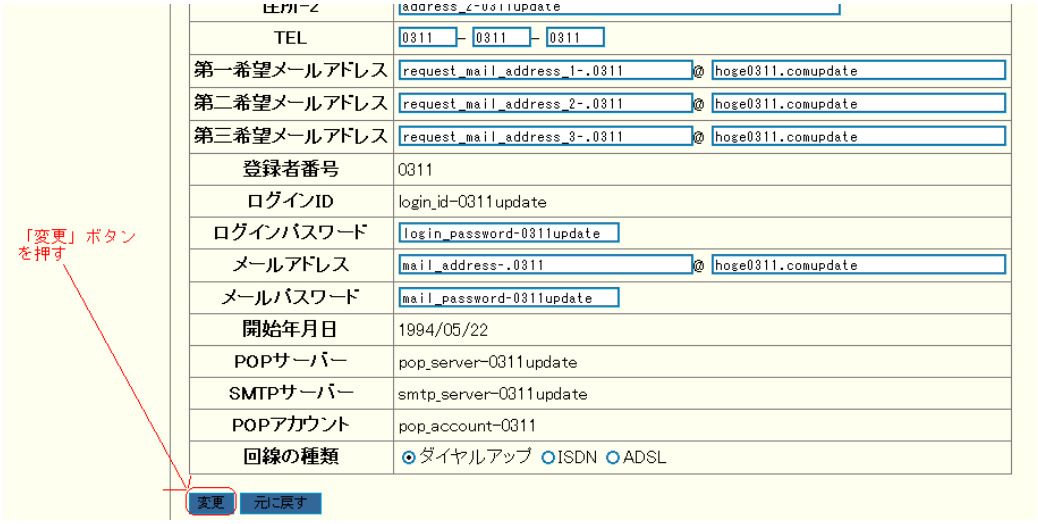
[1]　任意の登録者を選択します。



[2] 登録者編集画面が表示されます。登録者情報を変更します。ここでは仮に名前を「梨 名塩」、郵便番号を「100-1100」に変更します。



[3] 画面の下にある「変更」ボタンを押します。



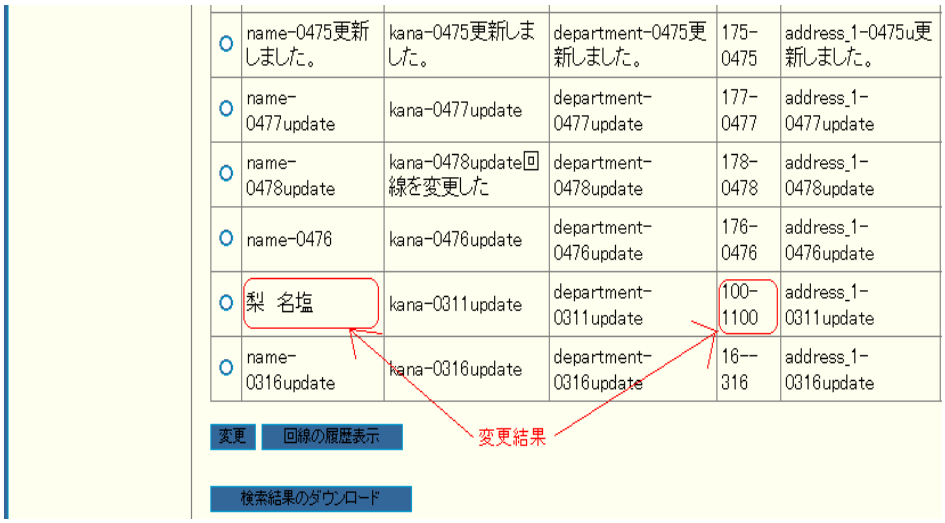
[4]　確認画面が表示され、変更内容を確認後「OK」ボタンを押します。

項目名	変更前	変更後
名前	name-0311update	梨 名塩
フリガナ	kana-0311update	kana-0311update
部署	department-0311update	department-0311update
郵便番号	11-0311	100-1100
住所1	address_1-0311update	address_1-0311update
住所2	address_2-0311update	address_2-0311update
電話番号	0311-0311-0311update	0311-0311-0311
第一希望メールアドレス	request_mail_address_1-0311@hoge0311.comupdate	request_mail_address_1-0311@hoge0311.comupdate
第二希望メールアドレス	request_mail_address_2-0311@hoge0311.comupdate	request_mail_address_2-0311@hoge0311.comupdate
第三希望メールアドレス	request_mail_address_3-0311@hoge0311.comupdate	request_mail_address_3-0311@hoge0311.comupdate
ログインパスワード	login_password-0311update	login_password-0311update
メールアドレス	mail_address-0311@hoge0311.comupdate	mail_address-0311@hoge0311.comupdate
メールパス	mail_password-0311update	mail_password-0311update
回線の種類	D	D

OK
キャンセル
「OK」ボタンを押す

[5]　完了画面が表示されます。

[6]　登録者情報が変更されました。



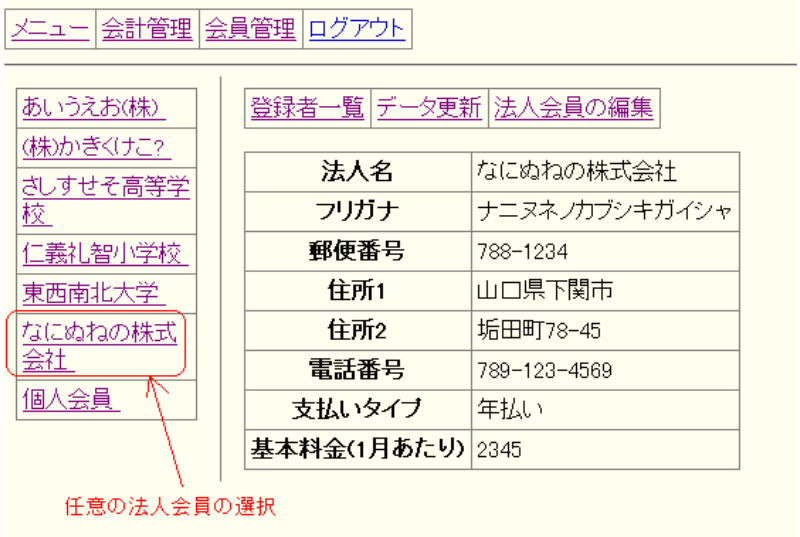
(イ) CSV を用いたデータ更新

CSV 形式のファイルを読み込むことができます。データ更新では更新モードを選択して「登録者の追加」、「登録者の変更」、「登録者の削除」を一括して行うことができます。デフォルトの更新モードは「登録者の追加」です。

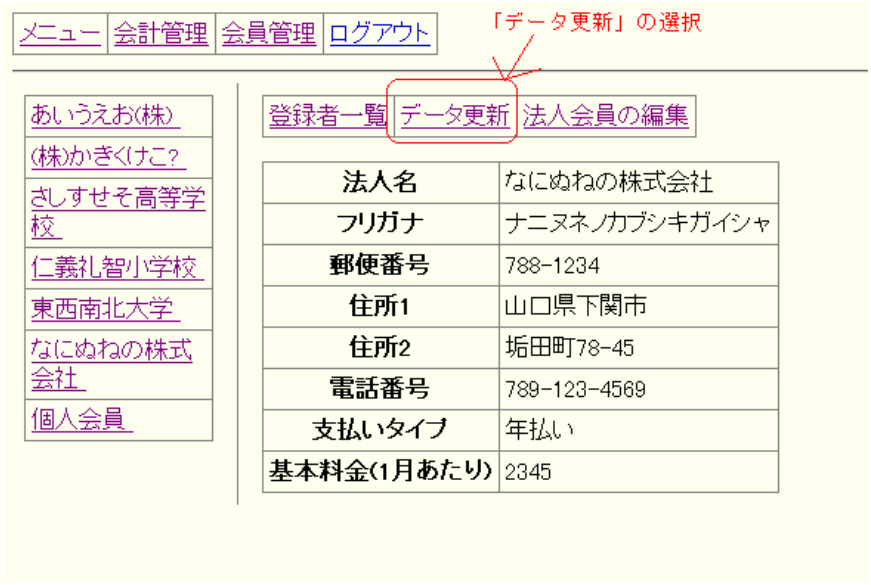
1. 登録者の追加

システムが CSV ファイルを読み込み、登録者のデータ更新を一括して行うことができます。登録者の追加の場合、同じ登録者番号を持っている登録者を再度追加することはできません。CSV ファイルの中に同じ登録者番号を持った登録者情報があった場合、データベースエラーが発生します。その場合、CSV にある全ての登録者は更新されません。

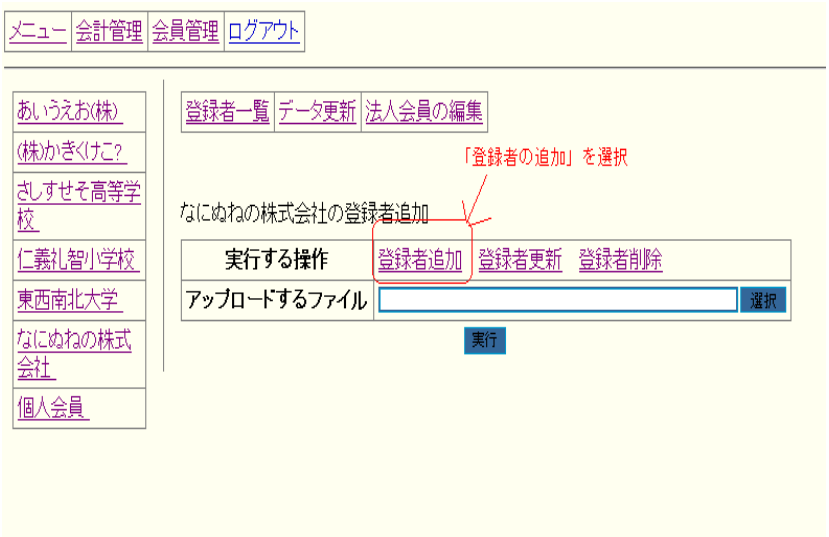
[1]　会員管理画面が表示されている状態で任意の法人を選択します。



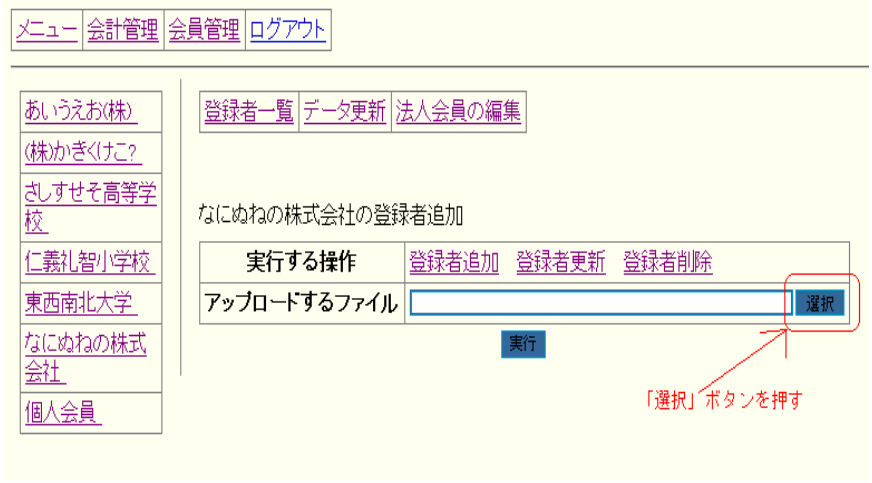
[2]　「データ更新」リンクをクリックします。



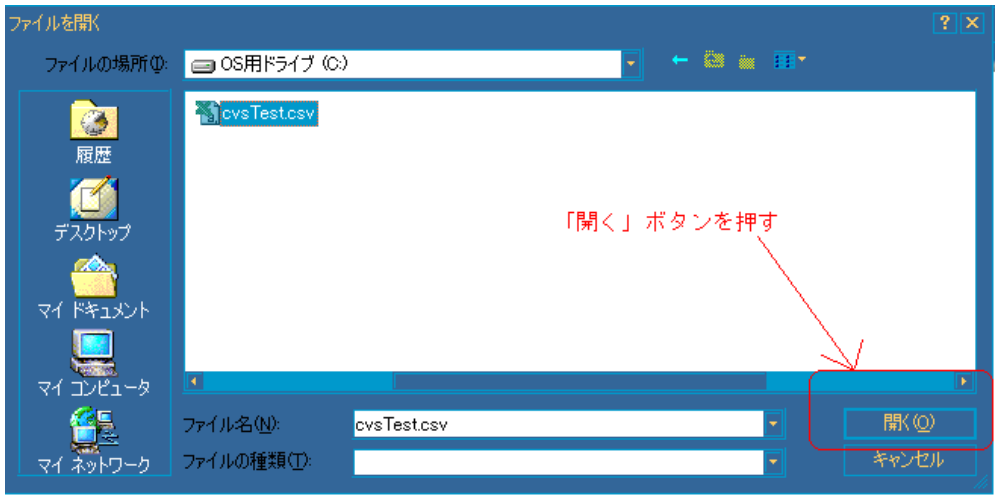
[3]　更新モードを「登録者の追加」に設定します。



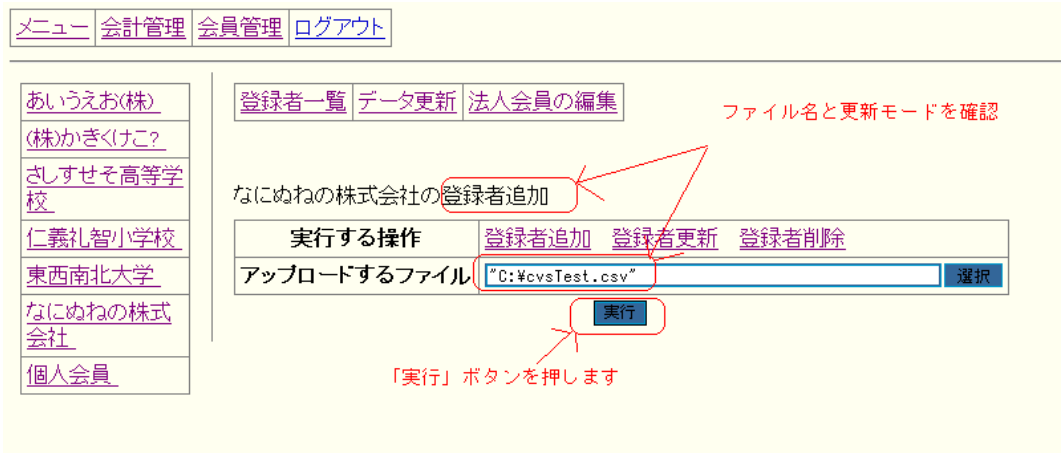
[4]　「選択」ボタンを押します。



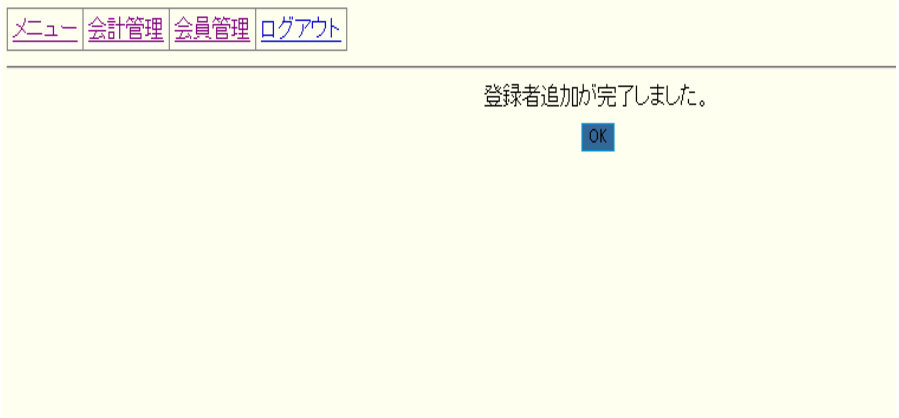
[5]　CSV ファイルを選択し、「開く」ボタンを押します。



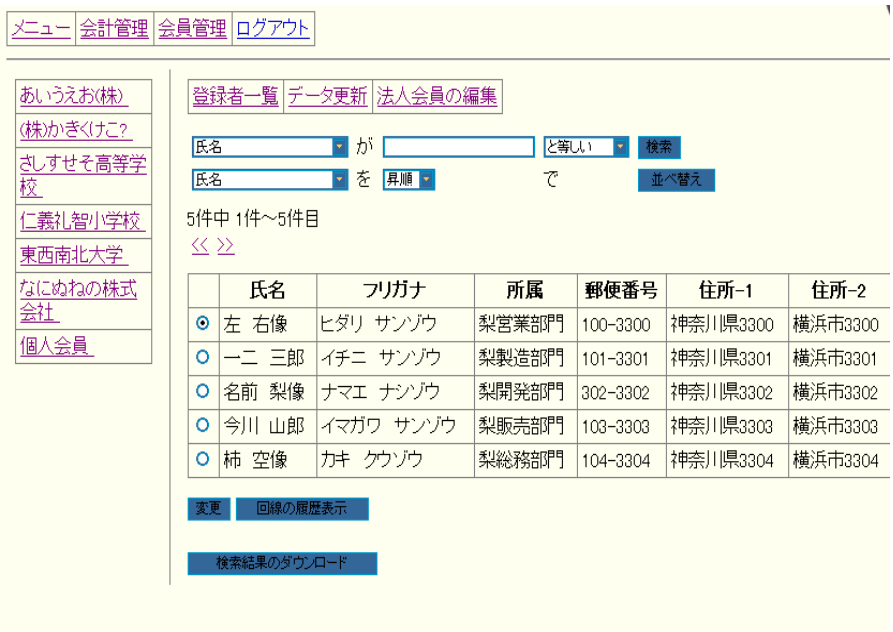
[6]　選択中の法人会員と更新モードを確認し、「実行」ボタンを押します。



[7]　完了画面が表示されます。



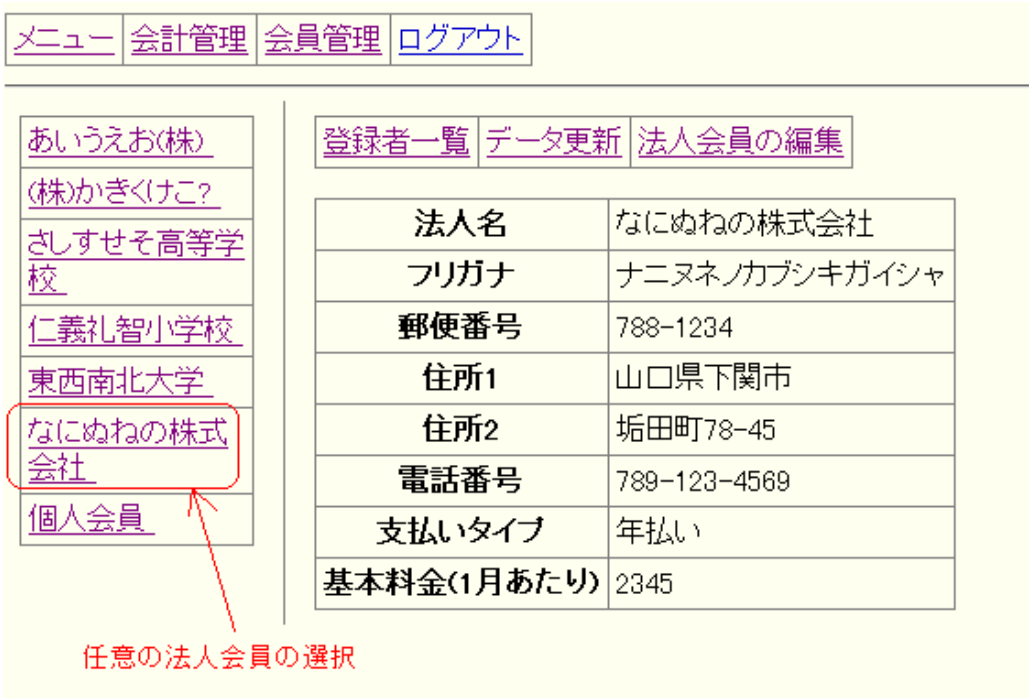
[8]　登録者情報が追加されました。



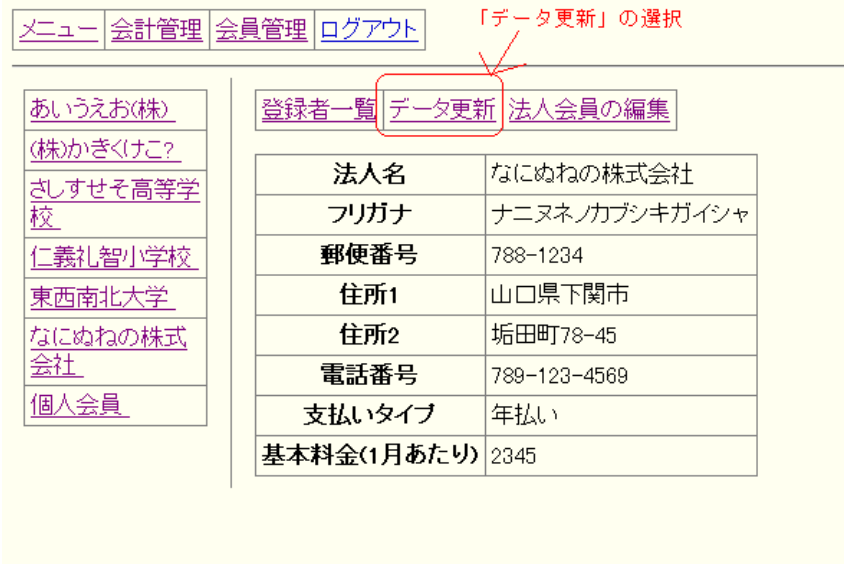
2. 登録者の更新

システムが CSV ファイルを読み込み、登録者のデータ更新を一括して行うことができます。ここでは仮に全ての名前に「?」、全てのフリガナに「+」を付け加えた CSV ファイルを読み込む手順をみていきます。

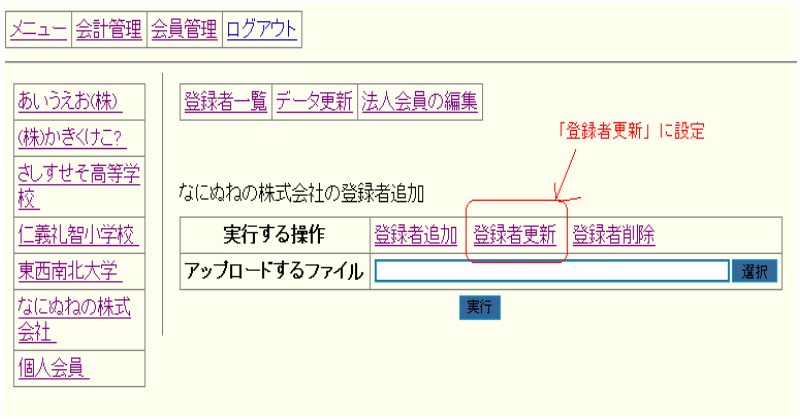
[1] 会員管理画面が表示されている状態で任意の法人を選択します。



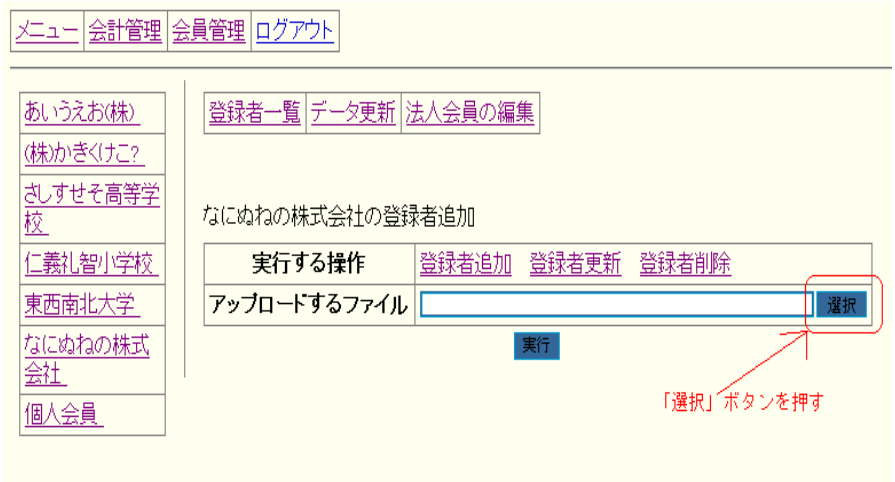
[2] 「データ更新」リンクをクリックします。



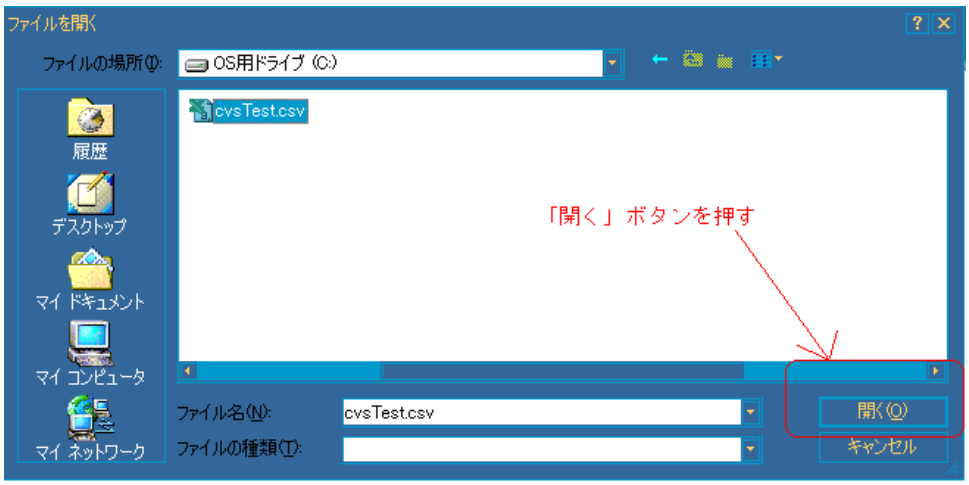
[3]　更新モードを「登録者更新」に設定します。



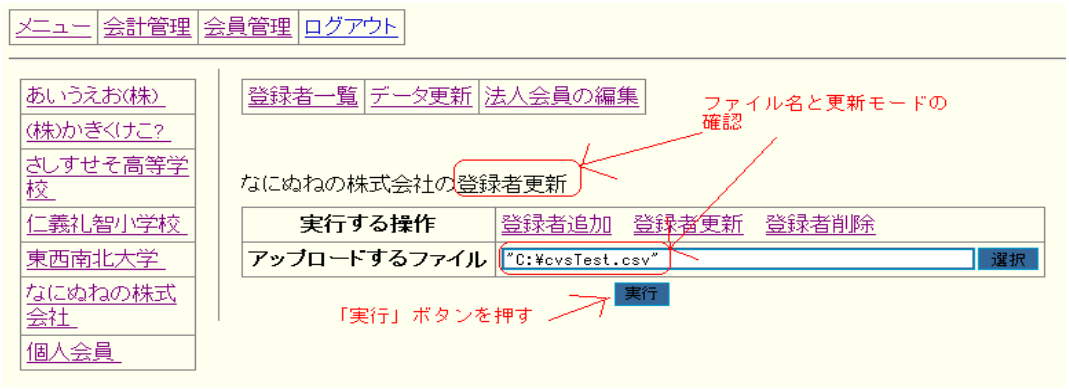
[4]　「選択」ボタンを押します。



[5]　CSV ファイルを選択し、「開く」ボタンを押します。

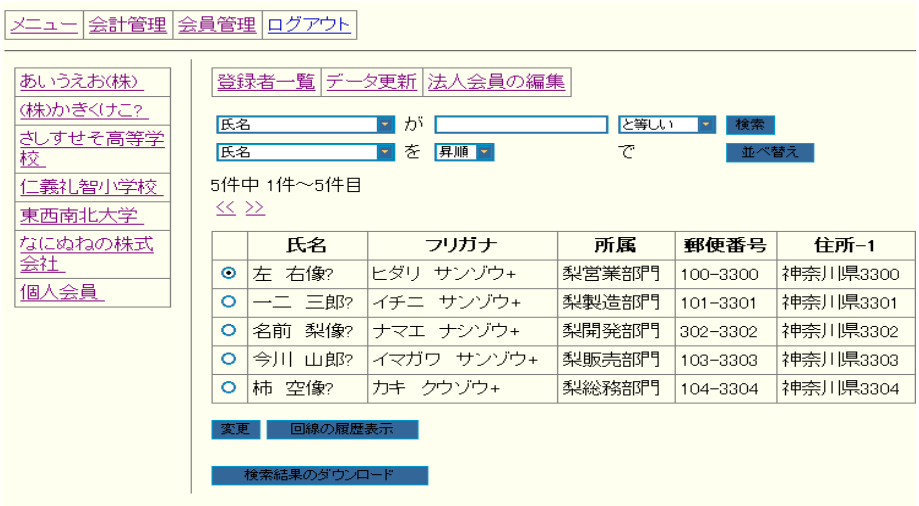


[6]　選択中の法人会員と更新モードを確認し、「実行」ボタンを押します。



[7]　確認画面が表示されます。

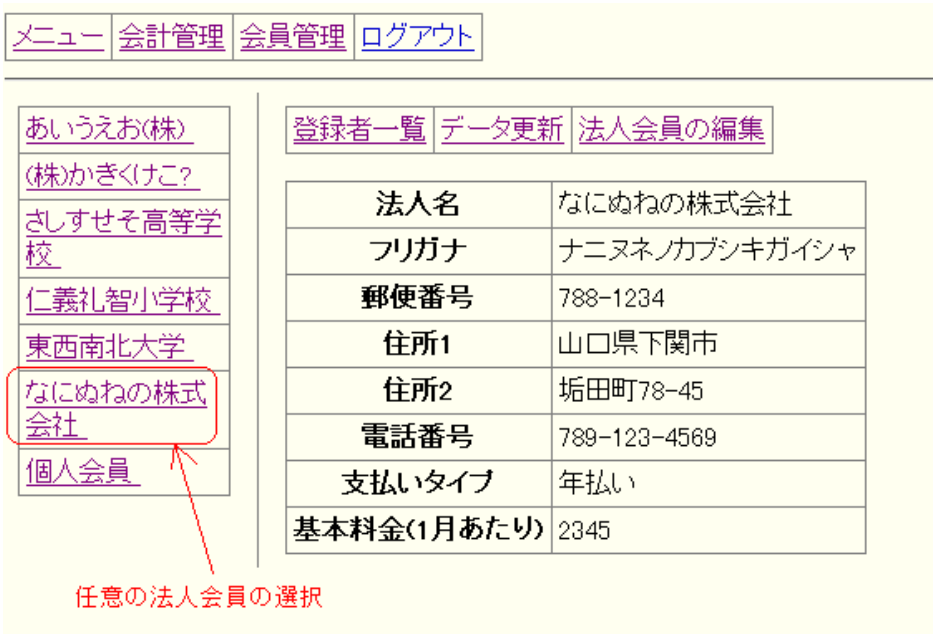
[8]　登録者情報が更新されました。



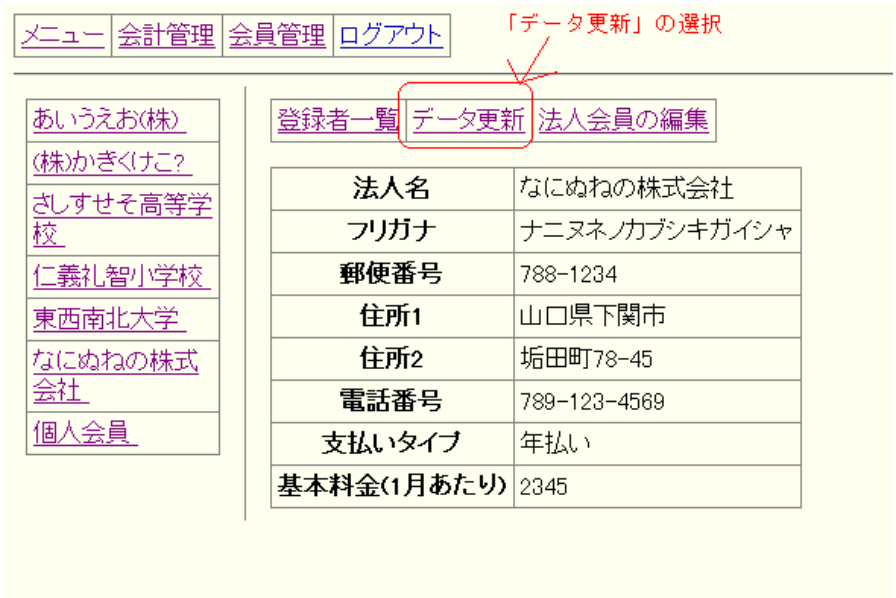
3. 登録者の削除

登録者の一括削除を行います。この機能では登録者番号をキーにして、登録者の一括削除を行っています。ではここでは仮に「左　右像」さんと「一二　三像」さんを削除してみましょう。また CSV ファイルで、名前が違っていたとしても登録者番号さえ一致していれば、登録者は削除されます。

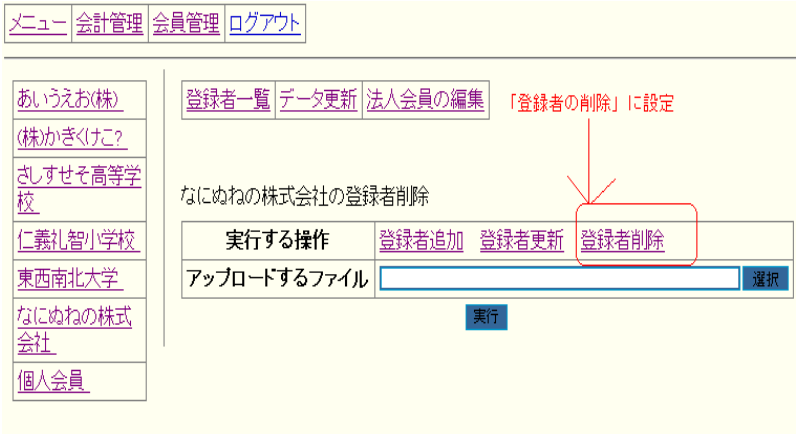
[1] 会員管理画面が表示されている状態で任意の法人を選択します。



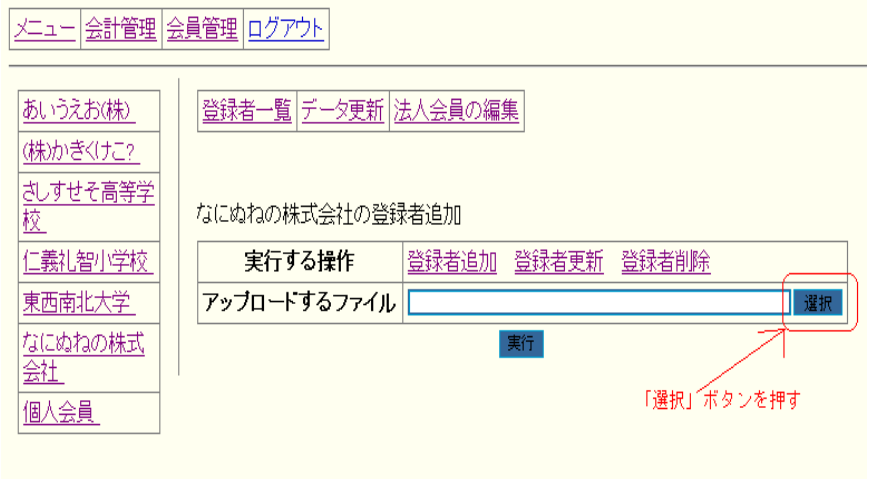
[2] 「データ更新」リンクをクリックします。



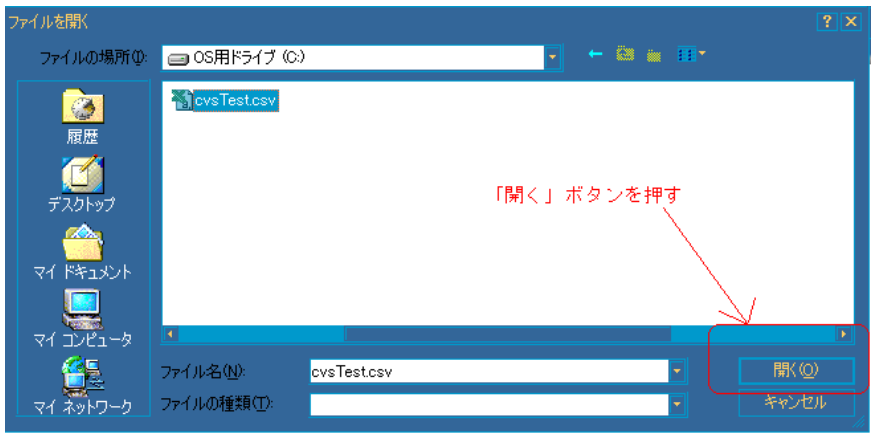
[3]　更新モードを「登録者削除」に設定します。



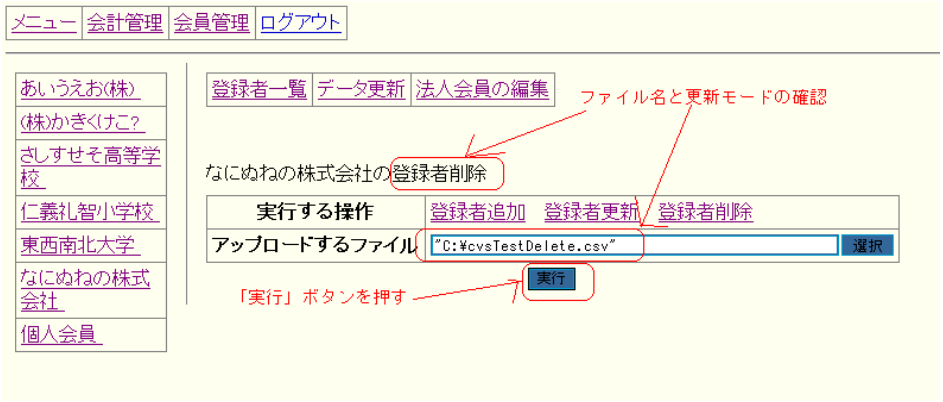
[4]　「選択」ボタンを押します。



[5]　CSV ファイルを選択し、「開く」ボタンを押します。



[6]　選択中の法人会員と更新モードを確認し、「実行」ボタンを押します。



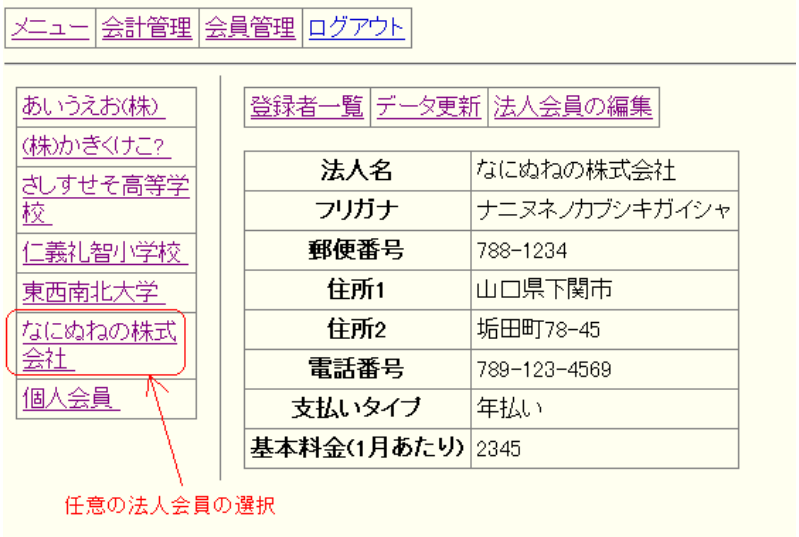
[7]　完了画面が表示されます。

[8]　登録者情報が削除されました。

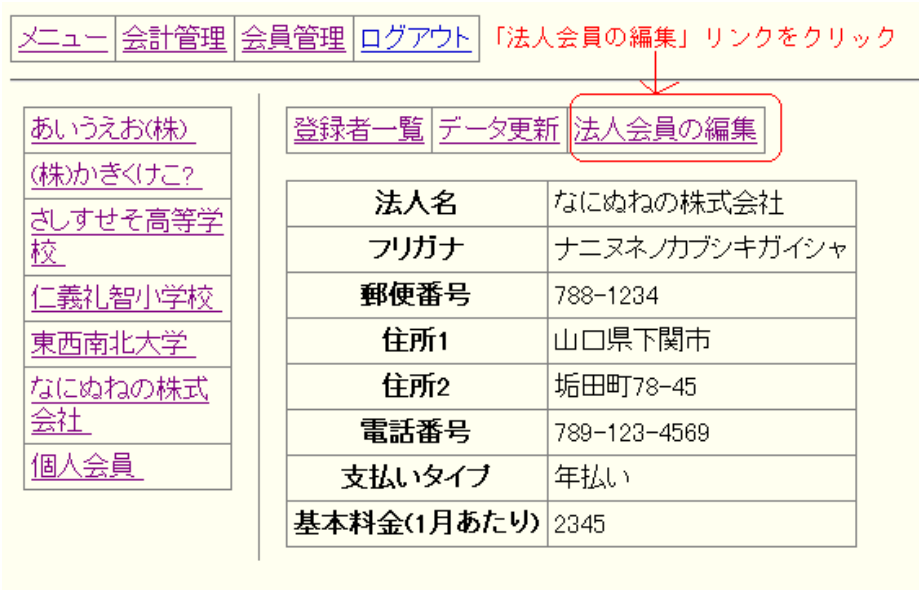
(ウ) 法人会員情報の変更

法人会員の情報を変更します。ここでの変更は会計管理の方にも変更が反映されます。

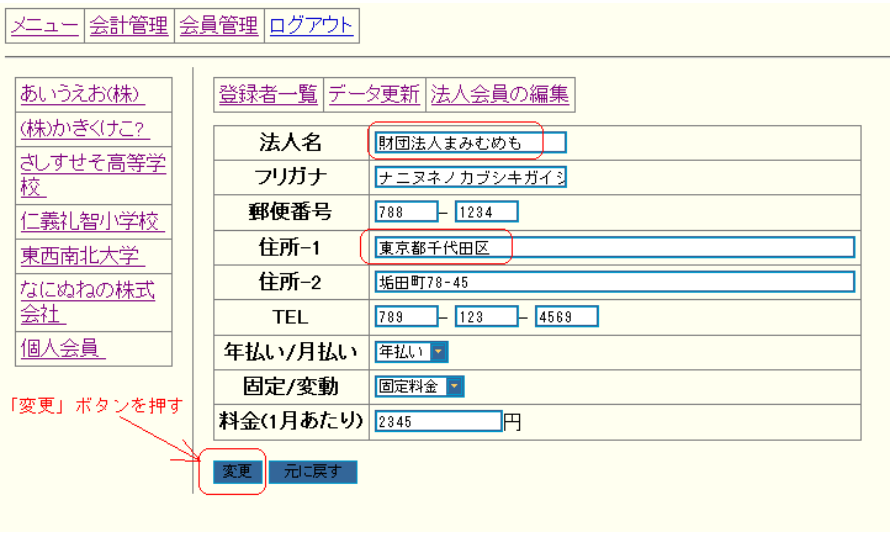
[1]　情報を変更したい法人会員を選択します。



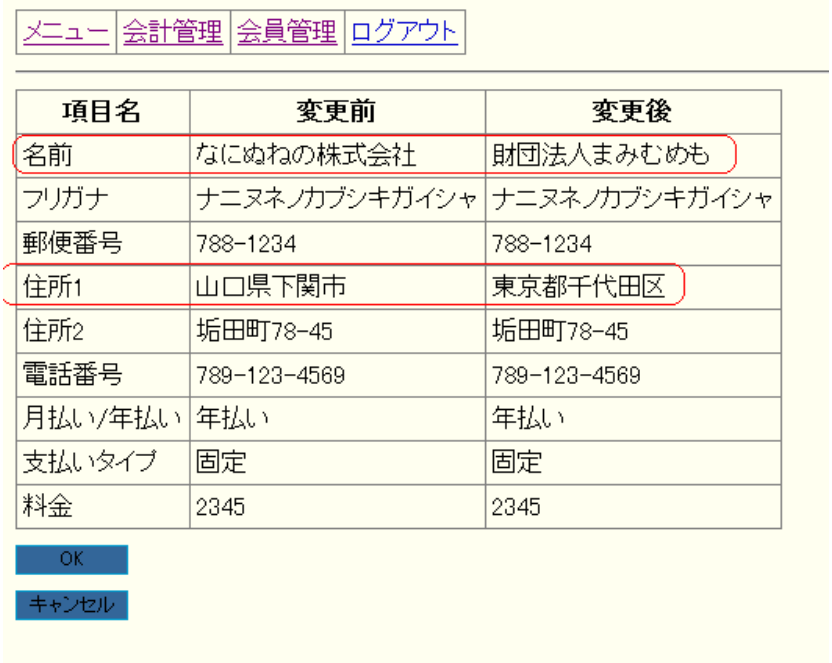
[2]　「法人会員の編集」リンクをクリックします。



[3] 法人編集画面が表示されるので、ここでは名前を「財団法人まみむめも」、住所-1 を「東京都千代田区」にしてみましょう。入力後、「変更」ボタンを押します。

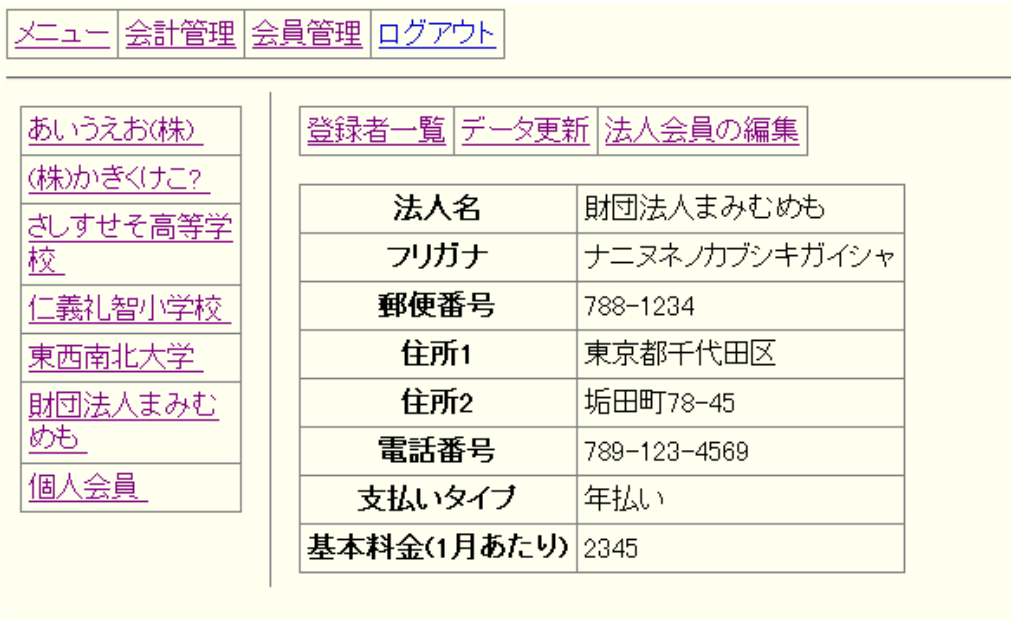


[4] 確認画面が表示されます。変更された情報を確認後、「OK」ボタンを押します。



[5]　完了画面が表示されます。

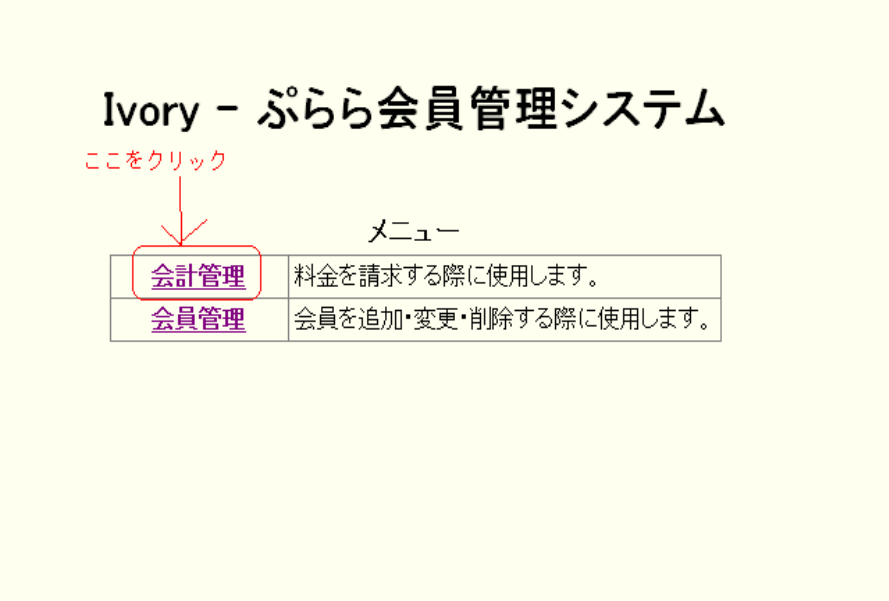
[6]　法人情報が変更されました。



3. 会計管理機能

会計管理は会員選択画面から操作を行います。この画面を表示すると法人会員と個人会員の一覧が表示されます。

[1] ログイン後、メニュー画面を表示します。「会計管理」リンクをクリックしてください。ログインの方法は会員管理に関する説明の部分をご覧ください。

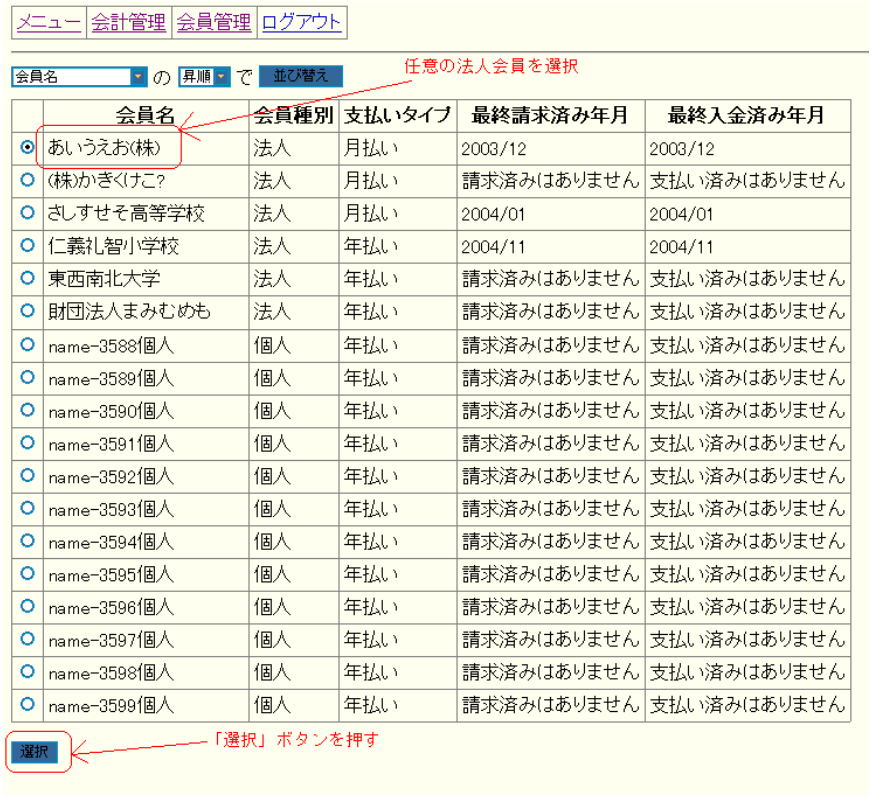


[2] 会員選択画面が表示されました。

	会員名	会員種別	支払いタイプ	最終請求済み年月	最終入金済み年月
◉	あいうえお(株)	法人	月払い	2003/12	2003/12
○	(株)かきくけこ	法人	月払い	請求済みはありません	支払い済みはありません
○	さしすせそ高等学校	法人	月払い	2004/01	2004/01
○	仁義礼智小学校	法人	年払い	2004/11	2004/11
○	東西南北大学	法人	年払い	請求済みはありません	支払い済みはありません
○	財団法人まみむめも	法人	年払い	請求済みはありません	支払い済みはありません
○	name-3300個人	個人	年払い	請求済みはありません	支払い済みはありません
○	name-3301個人	個人	年払い	請求済みはありません	支払い済みはありません
○	name-3302個人	個人	年払い	請求済みはありません	支払い済みはありません
○	name-3303個人	個人	年払い	請求済みはありません	支払い済みはありません
○	name-3304個人	個人	年払い	請求済みはありません	支払い済みはありません
○	name-3305個人	個人	年払い	請求済みはありません	支払い済みはありません
○	name-3306個人	個人	年払い	請求済みはありません	支払い済みはありません

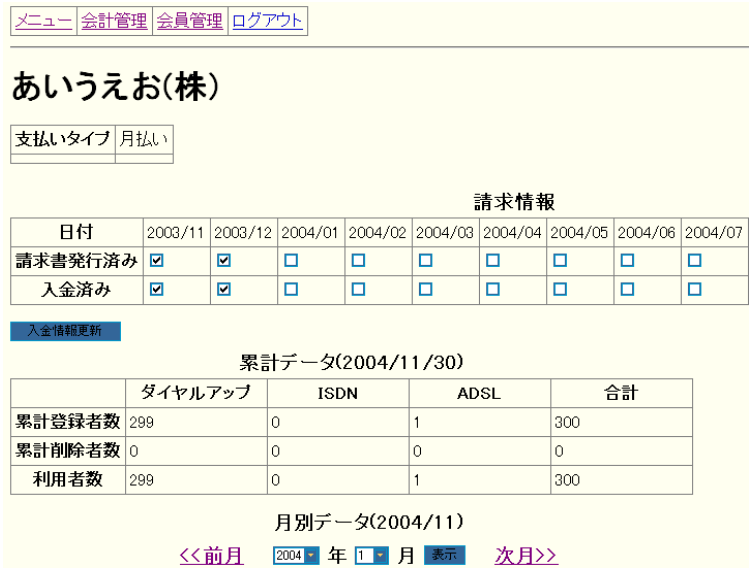
1. 法人の会計情報の閲覧

   法人会員の情報を閲覧します。会員選択画面で任意の法人会員を選択し、「選択」ボタンを押します。



	会員名	会員種別	支払いタイプ	最終請求済み年月	最終入金済み年月
◉	あいうえお(株)	法人	月払い	2003/12	2003/12
○	(株)かきくけこ?	法人	月払い	請求済みはありません	支払い済みはありません
○	さしすせそ高等学校	法人	月払い	2004/01	2004/01
○	仁義礼智小学校	法人	年払い	2004/11	2004/11
○	東西南北大学	法人	年払い	請求済みはありません	支払い済みはありません
○	財団法人まみむめも	法人	年払い	請求済みはありません	支払い済みはありません
○	name-3588個人	個人	年払い	請求済みはありません	支払い済みはありません
○	name-3589個人	個人	年払い	請求済みはありません	支払い済みはありません
○	name-3590個人	個人	年払い	請求済みはありません	支払い済みはありません
○	name-3591個人	個人	年払い	請求済みはありません	支払い済みはありません
○	name-3592個人	個人	年払い	請求済みはありません	支払い済みはありません
○	name-3593個人	個人	年払い	請求済みはありません	支払い済みはありません
○	name-3594個人	個人	年払い	請求済みはありません	支払い済みはありません
○	name-3595個人	個人	年払い	請求済みはありません	支払い済みはありません
○	name-3596個人	個人	年払い	請求済みはありません	支払い済みはありません
○	name-3597個人	個人	年払い	請求済みはありません	支払い済みはありません
○	name-3598個人	個人	年払い	請求済みはありません	支払い済みはありません
○	name-3599個人	個人	年払い	請求済みはありません	支払い済みはありません

[1] 法人会員の会計情報が表示されました。



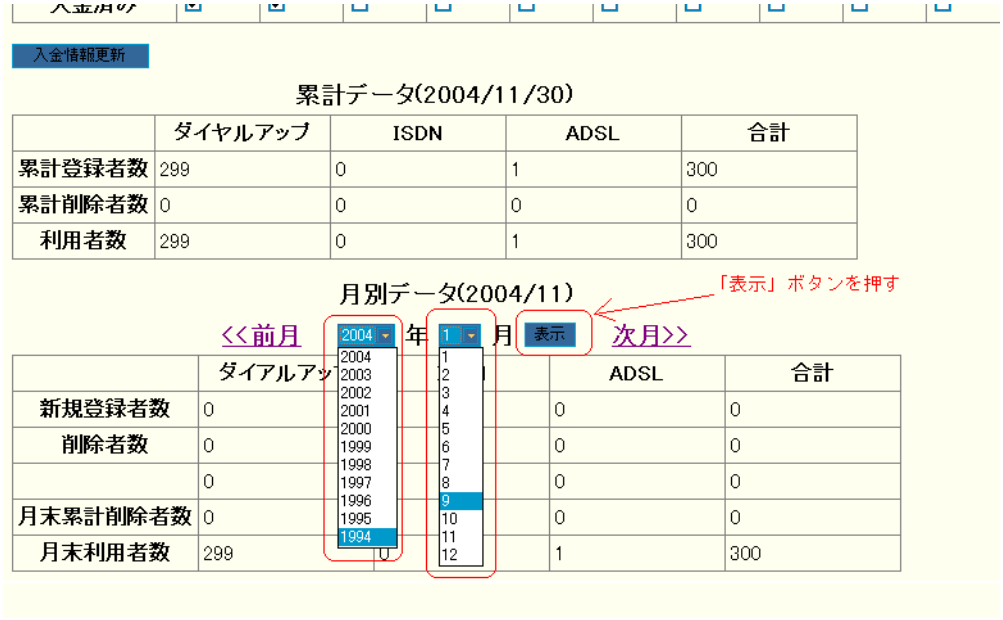
日付	2003/11	2003/12	2004/01	2004/02	2004/03	2004/04	2004/05	2004/06	2004/07
請求書発行済み	☑	☑	☐	☐	☐	☐	☐	☐	☐
入金済み	☑	☑	☐	☐	☐	☐	☐	☐	☐

	ダイヤルアップ	ISDN	ADSL	合計
累計登録者数	299	0	1	300
累計削除者数	0	0	0	0
利用者数	299	0	1	300

(ア) 月別データの表示

累計データを月別で表示します。

[1] 法人会計画面の画面下部の月別累計データのところで閲覧する年と月を選択し、「表示」ボタンを押します。ここでは 1994 年 9 月のデータを表示します。



[2] 1994 年 9 月のデータが表示されました。

請求書発行済み	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐
入金済み	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐

入金情報更新

累計データ(2004/11/30)

	ダイヤルアップ	ISDN	ADSL	合計
累計登録者数	299	0	1	300
累計削除者数	0	0	0	0
利用者数	299	0	1	300

月別データ(1994/09)

<<前月 2004 年 1 月 表示 次月>>

	ダイアルアップ	ISDN	ADSL	合計
新規登録者数	0	0	0	0
削除者数	0	0	0	0
月末累計登録者数	0	0	0	0
月末累計削除者数	0	0	0	0
月末利用者数	96	0	0	96

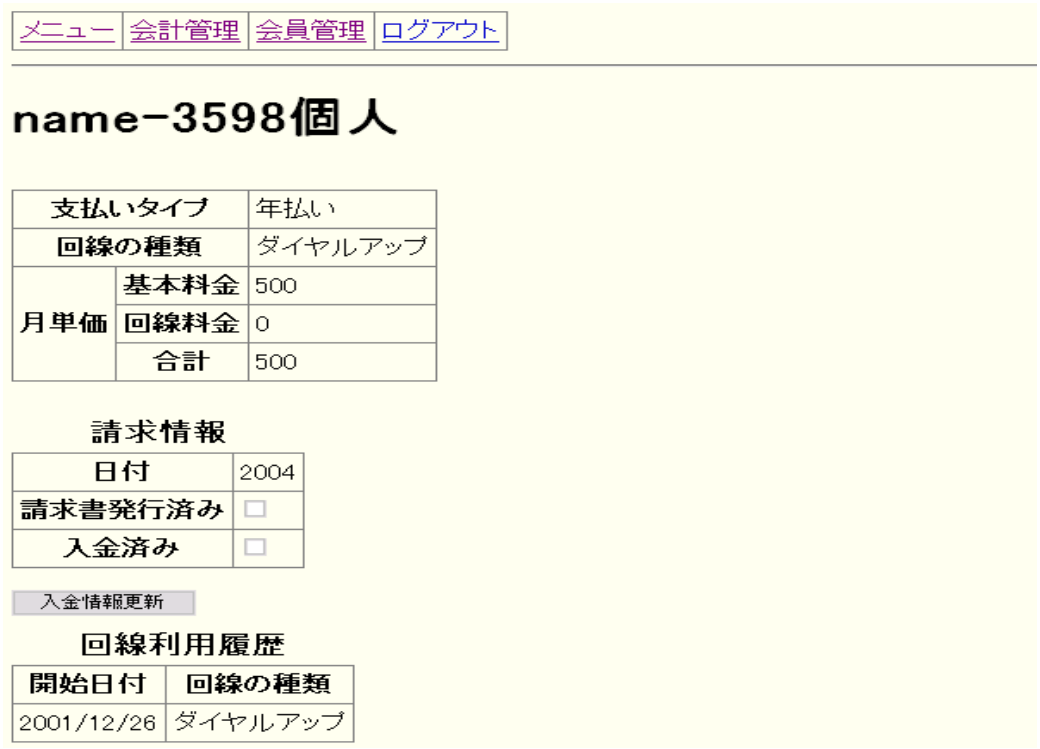
2. 個人の会計情報の閲覧

個人会員の会計情報を表示します。

[1] 会員管理画面で任意の個人会員を選択します。

	会員名	会員種別	支払いタイプ	最終請求済み年月	最終入金済み年月
○	あいうえお(株)	法人	月払い	2003/12	2003/12
○	(株)かきくけこ?	法人	月払い	請求済みはありません	支払い済みはありません
○	さしすせそ高等学校	法人	月払い	2004/01	2004/01
○	仁義礼智小学校	法人	年払い	2004/11	2004/11
○	東西南北大学	法人	年払い	請求済みはありません	支払い済みはありません
○	財団法人まみむめも	法人	年払い	請求済みはありません	支払い済みはありません
○	name-3588個人	個人	年払い	請求済みはありません	支払い済みはありません
○	name-3589個人	個人	年払い	請求済みはありません	支払い済みはありません
○	name-3590個人	個人	年払い	請求済みはありません	支払い済みはありません
○	name-3591個人	個人	年払い	請求済みはありません	支払い済みはありません
○	name-3592個人	個人	年払い	請求済みはありません	支払い済みはありません
○	name-3593個人	個人	年払い	請求済みはありません	支払い済みはありません
○	name-3594個人	個人	年払い	請求済みはありません	支払い済みはありません
○	name-3595個人	個人	年払い	請求済みはありません	支払い済みはありません
○	name-3596個人	個人	年払い	請求済みはありません	支払い済みはありません
○	name-3597個人	個人	年払い	請求済みはありません	支払い済みはありません
◉	name-3598個人	個人	年払い	請求済みはありません	支払い済みはありません
○	name-3599個人	個人	年払い	請求済みはありません	支払い済みはありません

[2]　個人会員の会計情報が表示されました。



4.　故障かなと思ったら

1. 症状と原因、そして解決策
   ①　データベースエラーが発生する
      1. 症状：データベースエラー画面が表示される。
      2. 原因：登録者番号が重複している。
      3. 解決策：登録者番号の重複をなくてください。
   ②　値チェックエラーが発生する。
      1. 症状：値チェックエラー画面が表示される。
      2. 原因：法人会員編集、登録者編集、CSV によるデータ更新を行う際に値の検査を行っており、その検査で不適切な値と判断されている。郵便番号や電話番号に数字以外の値が入っていないかチェックする。
      3. 解決策：不適切な値をチェックして再度入力を行ってください。

5. その他

(ア) 用語解説（あいうえお順）

① か行：

1. 会員（かいいん）：
   「法人会員」と「個人会員」の総称。
2. 回線（かいせん）：
   登録者が利用しているインターネット回線。ダイヤルアップ・ISDN・ADSL に共通する説明で用いる。
3. 回線変更履歴（かいせんへんこうりれき）：
   登録者が回線を変更した履歴。履歴は登録者一人の変更履歴と法人会員に所属する登録者の全履歴を閲覧できる。
4. 検索（けんさく）：
   登録者一覧画面において特定の条件の一致する登録者を探し出す機能。
5. 検索結果（けんさくけっか）：
   検索によって得られた登録者の一覧。
6. 検索する値（けんさくするあたい）：
   登録者を特定するための文字列、数値などの値。
7. 検索する状態（けんさくするじょうたい）：
   登録者を特定するための状態。今回実装された条件として「**と等しい**」「**を含む**」「**から始まる**」「**で終わる**」「**以上**」「**以下**」「**より上**」「**より下**」がある。また「名前」が「1234」「より上」というような条件を設定した場合エラーは発生せず、0 件の一致という検索結果が表示される。
8. 検索する列（けんさくするれつ）：
   検索を行う列項目。登録者一覧画面では一覧の一番上の「名前」「フリガナ」などがそれにあたる。
9. 個人会員（こじんかいいん）：
   AES を通して Plala に入会した登録者のこと。

② さ行：

1. 請求済み（せいきゅうずみ）：
   会員に対する請求書の発行が済んでいる状態。
2. 請求履歴（せいきゅうりれき）：
   会員に対する入金済みと請求済みを会員ごとの請求の間隔で区

切った履歴。入金と請求が終了した過去の履歴は削除される。

3. 絞込み検索（しぼりこみけんさく）：
   現在表示されている検索結果あるいは並べ替え結果を元にさらに詳細に検索を行うこと。登録者一覧画面で絞込み検索後、検索結果を最初の「現在選択中の法人に所属する全登録者の表示」に戻したい場合は登録者一覧画面の「登録者一覧」のリンクをクリックする。
4. 絞込み並べ替え（しぼりこみならべかえ）：
   現在表示されている検索結果あるいは並べ替え結果を元にさらに詳細に並べ替えを行うこと。登録者一覧画面で絞込み並べ替え後、検索結果を最初の「現在選択中の法人に所属する全登録者の表示」に戻したい場合は登録者一覧画面の「登録者一覧」のリンクをクリックする。

③ た行：

1. データ更新（でーたこうしん）：
   CSV ファイル形式のファイルを用いて登録者情報を一括で更新すること。理論上無限のデータを一括で更新することができるが、更新処理時間はデータ量に比例する。
2. 登録者（とうろくしゃ）：
   ぷららに登録している人。
3. 登録者番号（とうろくしゃばんごう）：
   全ての登録者が保持している、登録者を一意に特定するための番号。

④ な行：

1. 梨（なし）：
   このシステムのシンボルが梨なのは開発者の趣味がお菓子作りで、シンボルを考えるとき、ちょうど梨のお菓子を考えていたからである。
2. 並べ替え（ならべかえ）：
   特定の条件にしたがって一覧を整列させること。並べ替えで選択する条件は「並べ替えする列」と「並べ替えする順」。
3. 並べ替え結果（ならべかえけっか）：
   並べ替えを行った後、表示される並べ替えの一覧。
4. 並べ替えする列（ならべかえするれつ）：
   並べ替えを行うための列。「名前」、「郵便番号」まどの列。
5. 並べ替えする順（ならべかえするじゅん）：

昇順あるいは降順という順序。
昇順は小さいものが先頭。
降順は大きいものが先頭。
例）1.5.2.4.3.6 という数列を昇順、降順に並べ替える。
昇順：1.2.3.4.5.6
降順：6.5.4.3.2.1

6. 入金済み（にゅうきんずみ）：
会員の入金が済んでいる状態。

⑤ は行：

1. 法人会員（ほうじんかいいん）：
法人窓口を通して登録者になった者はその法人会員単位で管理される。

⑥ 英語：

1. CSV ファイル形式（しーえすぶいふぁいるけいしき）：
特定のデータの単位をカンマで区切る形式のファイル。表計算を行うソフトの中でもっとも標準的なデータ形式。
2. Ebony（えぼにー）：
黒檀という色の英語名。Ivory（象牙色）の補色。データベースサーバーの名前。アプリケーションサーバーである Ivory（象牙色）と対をなすという意味で名づけた。
3. Ivory（あいぼりー）：
象牙色という色の英語名。アプリケーションサーバーの名前。このシステムの開発者は元々「藍色」が好きで、「indigo」という名前を本当は付けようと考えていたが、「Ivory」が藍色と勘違いして、「Ivory」と付けた。その後、別の開発者に間違えを指摘されるまで様々なところで、この名前の設定を行っていため取り返しがつかなくなり、Ivory と名づけた。

(イ) 法人会員の追加と削除

法人の追加と削除はシステムに対して多大な影響を及ぼすため、通常の利用者は利用できないように設定しました。法人の追加と削除はメンテナンス担当者との相談の上、処理を行ってください。

(ウ) バグの報告

万が一不具合が発生した場合はメンテナンス担当者にご報告ください。